

M U N I X

COMMANDS

VOLUME I a

Information in this document is subject to change without notice and does not represent a commitment on the part of Periphäre Computer Systeme GmbH. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement.

Doc.-No.: D930100V1ae0484

Trademarks: MUNIX for PCS
DEC, PDP for DEC
UNIX for Bell Laboratories

Copyright 1984 by
PCS GmbH, Pfaelzer-Wald-Strasse 36, D-8000 Muenchen 90, tel. 089/ 67804-0
The information contained herein is the property of PCS and shall neither be reproduced in whole or in part without PCS's prior written approval nor be implied to grant any licence to make, use, or sell equipment manufactured in accordance herewith.

PCS reserves the right to make changes without notice in the specifications and materials contained herein and shall not be responsible for any damages (including consequential) caused by reliance on the materials presented.

Copyright 1979, Bell Telephone Laboratories, Incorporated.
Holders of a UNIXTM software license are permitted to copy this document, or any portion of it, as necessary for licensed use of the software, provided this copyright notice and statement of permission are included.

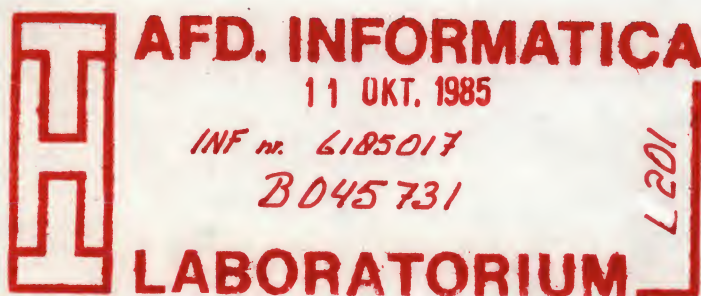
The MUNIX- documentation is divided in 3 main parts, which are currently provided in 4 binders. The corresponding binder is shown in bold face below.

MUNIX I COMMANDS, PROGRAMMING, SPECIALS, MAINTENANCE

MUNIX I a **COMMANDS**
(binder 1) Introduction
 Release-Notes
 How to get started
 Table of contents
 Permuted Index
 Commands and Application Programms
 in alphabetic order

MUNIX I b **PROGRAMMING, SPECIALS, MAINTENANCE**
(binder 2) System calls
 Subroutines
 Special files
 File Formats
 Games
 Miscellaneous Facilities
 Maintenance

MUNIX II **TUTORIALS**
(binder 3) USER'S GUIDE
 PROGRAMMING GUIDE
 SUPPORT TOOLS GUIDE
 DOCUMENT PROCESSING GUIDE
 UNIX System V Init and Getty
 UUCP Tutorial
 UUCP Administrator's Manual
 UNIX System Accounting
 File System Checking
 LP Spooling System
 UNIX System Remote Job Entry



1. The following information is being furnished to you for your information only. It is not to be used for any other purpose.

STATEMENT OF INFORMATION

Page 1

1. The following information is being furnished to you for your information only. It is not to be used for any other purpose.

Page 2

2. The following information is being furnished to you for your information only. It is not to be used for any other purpose.

Page 3

3. The following information is being furnished to you for your information only. It is not to be used for any other purpose.

Page 4

AND INFORMATION
LABORATORY

MUNIX III
(binder 4)

BASICS, OPTIONS
Current MUNIX Commands - Summary
Setting up MUNIX V1.5
UNIX Programming
Assembler Reference Manual
Assembler 68000 User's Guide - outdated
A Portable Fortran 77 Compiler
A Tutorial Introduction to ADB
CADMUS Testmonitor V1.0
Bad Block Behandlung / Bad Sector Handling
Typing Documents on the UNIX System
Using the -MS Macros
The UNIX Time-Sharing System
UNIX Implementaion
The UNIX I/O System
Display Editing with VI
Edit: A Tutorial
Ex Reference Manual
MED, PASCAL
Berkeley Extension Package

Options:

THE UNIVERSITY OF CHICAGO
DIVISION OF THE PHYSICAL SCIENCES
DEPARTMENT OF CHEMISTRY
530 SOUTH EAST ASIAN AVENUE
CHICAGO, ILLINOIS 60607
TEL. 373-5500
FAX 373-5500
WWW.CHEM.UCHICAGO.EDU

1997

1997

Introduction,

1

Release-Notes

2

How to get started

3

Table of contents

4

Permuted Index

5

Commands and Application programmes
in alphabetic order

6

7

8

9

10

Foreword

Dear reader, you now have a new version of the MUNIX User's Manual in your hands. We have put a great effort in it, proofread it many times, and run several checkers on it to verify the consistency. But ... If you find any errors, misspelling, or any kind of strange things, please let us know it. Criticism and suggestions to improve this documentation are heavily welcomed. PCS and, of course, the users of CADMUS Systems will benefit from your reaction.

We wish you a lot of pleasure with PCS products.

BL
DK
UH

ACKNOWLEDGEMENTS

The form and much of the content of this manual come from the *UNIX Programmer's Manual—Seventh Edition* (Volume 1), developed by M. D. McIlroy. In addition, parts of the present manual's contents are descended from the *UNIX Programmer's Manual—Sixth Edition* by K. Thompson and D. M. Ritchie (Bell Laboratories, May 1975), the *UNIX/TS User's Manual—Edition 1.1* by T. A. Dolotta and S. B. Olsson, eds. (Bell Laboratories, Jan. 1979), and the *PWB/UNIX User's Manual—Release 2.0* (Bell Laboratories, June 1979). P. E. Cannata and G. C. Vogel rewrote Section 2 for this edition. Many members of Centers 127 and 135, and of Laboratory 364 helped in the creation of this volume; their help is hereby gratefully acknowledged.

Murray Hill, New Jersey

T.A.D.
S.B.O.
A.G.P.

1. The first part of the report deals with the general situation of the country and the progress of the work during the year. It is divided into two main sections: the first section deals with the general situation of the country and the progress of the work during the year, and the second section deals with the results of the work during the year.

2. The second part of the report deals with the results of the work during the year. It is divided into two main sections: the first section deals with the results of the work during the year, and the second section deals with the results of the work during the year.

3. The third part of the report deals with the results of the work during the year. It is divided into two main sections: the first section deals with the results of the work during the year, and the second section deals with the results of the work during the year.

4. The fourth part of the report deals with the results of the work during the year. It is divided into two main sections: the first section deals with the results of the work during the year, and the second section deals with the results of the work during the year.

INTRODUCTION

This manual describes the features of UNIX. It provides neither a general overview of UNIX (for that, see "The UNIX Time-Sharing System," *BSTJ*, Vol. 57, No. 6, Part 2, pp. 1905-29, by D. M. Ritchie and K. Thompson), nor details of the implementation of the system (see "UNIX Implementation," *BSTJ*, same issue, pp. 1931-46).

Not all commands, features, and facilities described in this manual are available in every UNIX system; for example, *yacc*(1) is usually not available in a UNIX system running on a PDP-11/23. When in doubt, consult your system's administrator.

This manual is divided into eight sections, some containing inter-filed subclasses:

1. Commands and Application Programs:
 1. General-Purpose Commands.
 - 1B. Berkeley Extensions.
 - 1C. Communications Commands.
 - 1G. Graphics Commands.
 - 1M. System Maintenance Commands.
 - 1N. Newcastle Connection
2. System Calls.
3. Subroutines:
 - 3C. C and Assembler Library Routines.
 - 3F. Fortran Library Routines.
 - 3M. Mathematical Library Routines.
 - 3S. Standard I/O Library Routines.
 - 3X. Miscellaneous Routines.
4. Special Files.
5. File Formats.
6. Games.
7. Miscellaneous Facilities.
8. System Maintenance Procedures.

Section 1 (*Commands and Application Programs*) describes programs intended to be invoked directly by the user or by command language procedures, as opposed to subroutines, which are intended to be called by the user's programs. Commands generally reside in the directory */bin* (for binary programs). Some programs also reside in */usr/bin*, to save space in */bin*. These directories are searched automatically by the command interpreter called the *shell*. Sub-class 1C contains communication programs such as *cu*. These entries may differ from system to system. Sub-class 1M contains system maintenance programs such as *fsck*, *mkfs*, etc., which generally reside in the directory */etc*; these commands are not intended for use by the ordinary user due to their privileged nature. Some UNIX systems have a directory called */usr/local*, containing local commands.

Section 2 (*System Calls*) describes the entries into the UNIX supervisor, including the C language interface.

Introduction

Section 3 (*Subroutines*) describes the available subroutines. Their binary versions reside in various system libraries in the directories `/lib` and `/usr/lib`. See *intro*(3) for descriptions of these libraries and the files in which they are stored.

Section 4 (*Special Files*) discusses the characteristics of each system file that actually refers to an input/output device. The names in this section generally refer to the Digital Equipment Corporation's device names for the hardware, rather than to the names of the special files themselves.

Section 5 (*File Formats*) documents the structure of particular kinds of files; for example, the format of the output of the link editor is given in *a.out*(5). Excluded are files used by only one command (for example, the assembler's intermediate files). In general, the C language **struct** declarations corresponding to these formats can be found in the directories `/usr/include` and `/usr/include/sys`.

Section 6 (*Games*) describes the games and educational programs that, as a rule, reside in the directory `/usr/games`.

Section 7 (*Miscellaneous Facilities*) contains a variety of things. Included are descriptions of character sets, macro packages, etc.

Section 8 (*System Maintenance Procedures*) discusses crash recovery and boot procedures, etc. Information in this section is not of great interest to most users.

Each section consists of a number of independent entries of a page or so each. The name of the entry appears in the upper corners of its pages. Entries within each section are alphabetized, with the exception of the introductory entry that begins each section. The page numbers of each entry start at 1. Some entries may describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "major" name.

All entries are based on a common format, not all of whose parts always appear:

The **NAME** part gives the name(s) of the entry and briefly states its purpose.

The **SYNOPSIS** part summarizes the use of the program being described. A few conventions are used, particularly in Section 1 (*Commands*):

Boldface strings are literals and are to be typed just as they appear.

Italic strings usually represent substitutable argument prototypes and program names found elsewhere in the manual (they are underlined in the typed version of the entries).

Square brackets `[]` around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file", it always refers to a *file* name.

Ellipses ... are used to show that the previous argument prototype may be repeated.

Introduction

A final convention is used by the commands themselves. An argument beginning with a minus `-`, plus `+`, or equal sign `=` is often taken to be some sort of flag argument, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with `-`, `+`, or `=`.

The **DESCRIPTION** part discusses the subject at hand.

The **EXAMPLE(S)** part gives example(s) of usage, where appropriate.

The **FILES** part gives the file names that are built into the program.

The **SEE ALSO** part gives pointers to related information.

The **DIAGNOSTICS** part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

The **WARNINGS** part points out potential pitfalls.

The **BUGS** part gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

A table of contents and a permuted index derived from that table precede Section 1. On each *index* line, the title of the entry to which that line refers is followed by the appropriate section number in parentheses. This is important because there is considerable duplication of names among the sections, arising principally from commands that exist only to exercise a particular system call.

On most systems, all entries are available on-line via the *man*(1) command, q.v.

1944-1945

The first part of the report deals with the general situation in the country at the time of the survey. It is noted that the country was in a state of economic depression and that the population was suffering from widespread poverty and unemployment.

The second part of the report describes the methods used in the survey. It is noted that a series of interviews were conducted with a representative sample of the population. The interviews were conducted in a confidential and non-partisan manner.

The third part of the report presents the results of the survey. It is noted that the majority of the population was in favor of the proposed changes. The results also show that there was a strong feeling of loyalty to the country and its institutions.

The fourth part of the report discusses the implications of the survey results. It is noted that the results suggest that the proposed changes are likely to be accepted by the population. The results also suggest that there is a strong feeling of loyalty to the country and its institutions.

The fifth part of the report contains the conclusions of the survey. It is noted that the survey has shown that the population is in favor of the proposed changes and that there is a strong feeling of loyalty to the country and its institutions.

Please include Release-Notes here!

How To Get Started

HOW TO GET STARTED

This discussion provides the basic information you need to get started on UNIX: how to log in and log out, how to communicate through your terminal, and how to run a program. (See *UNIX for Beginners* by B. W. Kernighan for a more complete introduction to the system.)

Logging in. You must dial up UNIX from an appropriate terminal. UNIX supports full-duplex ASCII terminals. You must also have a valid user name, which may be obtained (together with the telephone number(s) of your UNIX system) from the administrator of your system. Common terminal speeds are 10, 15, 30, and 120 characters per second (110, 150, 300, and 1,200 baud); occasionally, speeds of 240, 480, and 960 characters per second (2,400, 4,800, and 9,600 baud) are also available. On some UNIX systems, there are separate telephone numbers for each available terminal speed, while on other systems several speeds may be served by a single telephone number. In the latter case, there is one "preferred" speed; if you dial in from a terminal set to a different speed, you will be greeted by a string of meaningless characters (the login: message at the wrong speed). Keep hitting the "break" or "attention" key until the login: message appears. Hard-wired terminals usually are set to the correct speed.

Most terminals have a speed switch that should be set to the appropriate speed and a half-/full-duplex switch that should be set to full-duplex. When a connection (at the speed of the terminal) has been established, the system types login: and you then type your user name followed by the "return" key. If you have a password (and you should!), the system asks for it, but does not print ("echo") it on the terminal. After you have logged in, the "return", "new-line", and "line-feed" keys will give exactly the same result.

It is important that you type your login name in lower case if possible; if you type upper-case letters, UNIX will assume that your terminal cannot generate lower-case letters and that you mean all subsequent upper-case input to be treated as lower case. When you have logged in successfully, the shell will type a **\$** to you. (The shell is described below under *How to run a program*.)

For more information, consult *login*(1) and *getty*(8), which discuss the login sequence in more detail, and *stty*(1), which tells you how to describe the characteristics of your terminal to the system (*profile*(5) explains how to accomplish this last task automatically every time you log in).

Logging out. There are two ways to log out:

1. You can simply hang up the phone.
2. You can log out by typing an end-of-file indication (ASCII EOT character, usually typed as "control-z") to the shell. The shell will terminate and the login: message will appear again.

How to communicate through your terminal. When you type to UNIX, a gnome deep in the system is gathering your characters and saving them. These characters will not be given to a program until you type a "return" (or "new-line"), as described above in *Logging in*.

UNIX terminal input/output is full-duplex. It has full read-ahead, which means

that you can type at any time, even while a program is typing at you. Of course, if you type during output, the output will have interspersed in it the input characters. However, whatever you type will be saved and interpreted in the correct sequence. There is a limit to the amount of read-ahead, but it is generous and not likely to be exceeded unless the system is in trouble. When the read-ahead limit is exceeded, the system throws away *all* the saved characters.

On an input line from a terminal, the character "control-x" "kills" all the characters typed before it. The character "backspace" erases the last character typed. Successive uses of "backspace" will erase characters back to, but not beyond, the beginning of the line. These default erase and kill characters can be changed; see *stty(1)*.

The ASCII DC3 (control-s) character can be used to temporarily stop output. It is useful with CRT terminals to prevent output from disappearing before it can be read. Output is resumed when a DC1 (control-q) or a second DC3 (or any other character, for that matter) is typed. The DC1 and DC3 characters are not passed to any other program when used in this manner.

The character "control-c" is not passed to programs, but instead generates an *interrupt signal*, just like the "break", "interrupt", or "attention" signal. This signal generally causes whatever program you are running to terminate. It is typically used to stop a long printout that you don't want. However, programs can arrange either to ignore this signal altogether, or to be notified when it happens (instead of being terminated). The editor *ed(1)*, for example, catches interrupts and stops what *it* is doing, instead of terminating, so that an interrupt can be used to halt an editor printout without losing the file being edited.

The *quit* signal is generated by typing the character "control-y". It not only causes a running program to terminate, but also generates a file with the "core image" of the terminated process. *Quit* is useful for debugging.

Besides adapting to the speed of the terminal, UNIX tries to be intelligent as to whether you have a terminal with the "new-line" function, or whether it must be simulated with a "carriage-return" and "line-feed" pair. In the latter case, all *input* "carriage-return" characters are changed to "line-feed" characters (the standard line delimiter), and a "carriage-return" and "line-feed" pair is echoed to the terminal. If you get into the wrong mode, the *stty(1)* command will rescue you.

Tab characters are used freely in UNIX source programs. If your terminal does not have the tab function, you can arrange to have tab characters changed into spaces during output, and echoed as spaces during input. Again, the *stty(1)* command will set or reset this mode. The system assumes that tabs are set every eight character positions. The *tabs(1)* command will set tab stops on your terminal, if that is possible.

How to run a program. When you have successfully logged into UNIX, a program called the shell is listening to your terminal. The shell reads the lines you type, splits them into a command name and its arguments, and executes the command. A command is simply an executable program. Normally, the shell looks

How To Get Started

first in your current directory (see *The current directory* below) for a program with the given name, and if none is there, then in system directories. There is nothing special about system-provided commands except that they are kept in directories where the shell can find them. You can also keep commands in your own directories and arrange for the shell to find them there.

The command name is the first word on an input line to the shell; the command and its arguments are separated from one another by space and/or tab characters.

When a program terminates, the shell will ordinarily regain control and type a **\$** at you to indicate that it is ready for another command. The shell has many other capabilities, which are described in detail in *sh(1)*.

The current directory. UNIX has a file system arranged in a hierarchy of directories. When the system administrator gave you a user name, he or she also created a directory for you (ordinarily with the same name as your user name, and known as your *login* or *home* directory). When you log in, that directory becomes your *current* or *working* directory, and any file name you type is by default assumed to be in that directory. Because you are the owner of this directory, you have full permissions to read, write, alter, or destroy its contents. Permissions to have your will with other directories and files will have been granted or denied to you by their respective owners, or by the system administrator. To change the current directory use *cd(1)*.

Path names. To refer to files not in the current directory, you must use a path name. Full path names begin with */*, which is the name of the *root* directory of the whole file system. After the slash comes the name of each directory containing the next sub-directory (followed by a */*), until finally the file name is reached (e.g., */usr/ae/filex* refers to file *filex* in directory *ae*, while *ae* is itself a subdirectory of *usr*; *usr* springs directly from the root directory). See *intro(2)* for a formal definition of *path name*.

If your current directory contains subdirectories, the path names of files therein begin with the name of the corresponding subdirectory (*without* a prefixed */*). Without important exception, a path name may be used anywhere a file name is required.

Important commands that modify the contents of files are *cp(1)*, *mv(1)*, and *rm(1)*, which respectively copy, move (i.e., rename), and remove files. To find out the status of files or directories, use *ls(1)*. Use *mkdir(1)* for making directories and *rmdir(1)* for destroying them.

For a fuller discussion of the file system, see the references cited at the beginning of the *INTRODUCTION* above. It may also be useful to glance through Section 2 of this manual, which discusses system calls, even if you don't intend to deal with the system at that level.

Writing a program. To enter the text of a source program into a UNIX file, use *ed(1)*. The four principal languages available under UNIX are C (see *cc(1)*), Fortran (see *f77(1)*), bs (a compiler/interpreter in the spirit of Basic, see *bs(1)*),

How To Get Started

and assembly language (see `as(1)`). After the program text has been entered with the editor and written into a file (whose name has the appropriate suffix), you can give the name of that file to the appropriate language processor as an argument. Normally, the output of the language processor will be left in a file in the current directory named `a.out` (if that output is precious, use `mv(1)` to give it a less vulnerable name). If the program is written in assembly language, you will probably need to load with it library subroutines (see `ld(1)`). Fortran and C call the loader automatically; programs written in `bs(1)` are interpreted and, therefore, do not need to be loaded.

When you have finally gone through this entire process without provoking any diagnostics, the resulting program can be run by giving its name to the shell in response to the `$` prompt.

If any execution (run-time) errors occur, you will need `adb(1)` to examine the remains of your program. On the VAX-11/780, a second debugger `sdb(1)`, which allows you to step through C statements rather than assembler instructions, is available.

Your programs can receive arguments from the command line just as system programs do; see `exec(2)`.

Text processing. Almost all text is entered through the editor `ed(1)`. The commands most often used to write text on a terminal are `cat(1)`, `pr(1)`, and `nroff(1)`. The `cat(1)` command simply dumps ASCII text on the terminal, with no processing at all. The `pr(1)` command paginates the text, supplies headings, and has a facility for multi-column output. `Nroff(1)` is an elaborate text formatting program, and requires careful forethought in entering both the text and the formatting commands into the input file; it produces output on a typewriter-like terminal. `Troff(1)` is very similar to `nroff(1)`, but produces its output on a phototypesetter (it was used to typeset this manual). There are several "macro" packages (especially the so-called `mm` package) that significantly ease the effort required to use `nroff(1)` and `troff(1)`; Section 7 entries for these packages indicate where you can find their detailed descriptions.

Surprises. Certain commands provide *inter-user* communication. Even if you do not plan to use them, it would be well to learn something about them, because someone else may aim them at you. To communicate with another user currently logged in, `write(1)` is used; `mail(1)` will leave a message whose presence will be announced to another user when he or she next logs in. The corresponding entries in this manual also suggest how to respond to these two commands if you are their target.

When you log in, a message-of-the-day may greet you before the first `$`.

Overview of the Different System Versions

We advise you to immediately complete this list with the upgrades and to fill the enclosed documents in your **MUNIX** binders and to remove the outdated documents as required.

Software	Version No.	Date	Remarks
MUNIX	1.5/01	03.84	delivered documentation
Typeset	V 1.0	22.04.83	delivered documentation
SCCS	V 1.0	22.04.83	delivered documentation
MED	V 3.0	03.84	delivered documentation
Berkeley	V 1.1	22.04.83	delivered documentation
PASCAL	V 2.2	9.09.83	delivered documentation

UNIT 1: THE HISTORY OF THE UNITED STATES

UNIT 1: THE HISTORY OF THE UNITED STATES
 This unit covers the early history of the United States, from the first settlers to the American Revolution. It includes a detailed study of the Pilgrims, the Puritans, and the early colonial period. The unit also explores the role of the British in the American Revolution and the impact of the war on the young nation.

Topic	Sub-Topic	Key Dates	Key Figures
Early Colonial Period	Pilgrims	1620	William Bradford
	Puritans	1630	John Winthrop
	Early Colonial Period	1607-1789	Various
American Revolution	Declaration of Independence	1776	Thomas Jefferson
	War of Independence	1775-1781	George Washington
	Constitution	1787	James Madison
Early National Period	Jeffersonian Democracy	1800-1820	Thomas Jefferson
	Monroe Doctrine	1823	James Monroe
	Early National Period	1800-1820	Various
Mid-19th Century	Manifest Destiny	1845	John O'Sullivan
	California Gold Rush	1849	James W. Wadsworth
	Mid-19th Century	1840-1860	Various
Civil War	Abolitionism	1840-1860	Frederick Douglass
	Civil War	1861-1865	Abraham Lincoln
	Reconstruction	1865-1877	Various
Late 19th Century	Industrial Revolution	1870-1900	Various
	Spanish-American War	1898	Various
	Late 19th Century	1870-1900	Various

TABLE OF CONTENTS

1. Commands and Application Programs

intro	introduction to commands
a68	MIT assembler
acctcom	search and print process accounting file(s)
adb	debugger
admin	create and administer SCCS files
apropos	locate commands by keyword lookup
ar	archive and library maintainer
as	assembler
at	execute commands at a later time
awk	pattern scanning and processing language
basename	strip filename affixes
basic	Basic Interpreter
bc	arbitrary-precision arithmetic language
bdiff	big diff
bfs	big file scanner
bs	a compiler/interpreter for modest-sized programs
cal	print calendar
calendar	reminder service
cat	concatenate and print files
cb	C program beautifier
cc	C compiler
cd	change working directory
cdc	change the delta commentary of an SCCS delta
checknews	check to see if user has news
checknr	check nroff/troff files
chfn	change full name of user
chmod	change mode
chown	change owner or group
clear	clear terminal screen
cmp	compare two files
col	filter reverse line feeds
colcrt	filter nroff output for CRT previewing
colrm	remove columns from a file
comb	combine SCCS deltas
comm	select or reject lines common to two sorted files
compact	compress and uncompress files, and cat them
cp	copy, link or move files
cpio	copy file archives in and out
cpp	the C language preprocessor
create	create a file
cref	make cross-reference listing
crypt	encode/decode
csh	a shell (command interpreter) with C-like syntax
csplit	context split
ctags	create a tags file
cu	call UNIX
cut	cut out selected fields of each line of a file
d	text database functions

Table of Contents

date	print and set the date
dbadd	add entry to an Emacs data base
dc	desk calculator
dd	convert and copy a file
delta	make a delta (change) to an SCCS file
deroff	remove nroff/troff, tbl, and eqn constructs
df	report number of free disk blocks
diction	print wordy sentences, interactive thesaurus for diction
diff	differential file comparator
diff3	3-way differential file comparison
dircmp	directory comparison
du	summarize disk usage
echo	echo arguments
ed	text editor
efl	Extended Fortran Language
emacs	a screen editor
env	set environment for command execution
eqn	format mathematical text for nroff or troff
error	analyze and disperse compiler error messages
ex	text editor
excr	run a program on another system
expand	expand tabs to spaces, and vice versa
expr	evaluate arguments as an expression
f77	Fortran 77 compiler
factor	factor a number
file	determine file type
find	find files
finger	user information lookup program
fmt	simple text formatter
fold	fold long lines for finite width output device
from	who is my mail from?
get	get a version of an SCCS file
getopt	parse command options
graph	draw a graph
grep	search a file for a pattern
head	give first few lines
help	ask for help
hp	handle special functions of HP 2640 and 2621-series terminals
hyphen	find hyphenated words
id	print user and group IDs and names
ifprolog	The prolog interpreter system
indent	indent and format a C program source
inews	submit news articles
join	relational database operator
kill	terminate a process
ld	loader
learn	computer aided instruction about UNIX
leave	remind you when you have to leave
lex	generator of lexical analysis programs
line	read one line
lint	a C program verifier

Table of Contents

lock	reserve a terminal
login	sign on
logname	get login name
look	find lines in a sorted list
lorder	find ordering relation for an object library
lpctrl	set options on the parallel line printer
lpr	line printer spooler
ls	list contents of directory
ltroff	troff to the CANON LBP or the Versatec V80
m4	macro processor
mail	send mail to users or read mail
mail	mail(1) using the Newcastle Connection instead of uucp.
make	maintain, update, and regenerate groups of programs
man	print entries in this manual
man	print sections of this manual
med	screen editor
mesg	permit or deny messages
mkdir	make a directory
mkstr	create an error message file by massaging C source
mm	print out documents formatted with the MM macros
mmchek	check usage of mm macros and eqn delimiters
mmt	typeset documents, view graphs, and slides
more	file perusal filter for crt viewing
msgs	system messages and junk mail program
mt	magnetic tape manipulating program
newgrp	log in to a new group
news	print news items
nice	run a command at low priority
nl	line numbering filter
nm	print name list
notes	a news system
od	octal dump
pack	packs or unpacks many files <---> one.
passwd	change login password
paste	merge same lines of several files or subsequent lines of one file
pc	Pascal compiler
plot	graphics filters
postnews	submit news articles
pr	print files
prep	prepare text for statistical processing
prmail	print out mail in the post office
prof	display profile data
prs	print an SCCS file
ps	process status
pti	phototypesetter interpreter
ptx	permuted index
pwd	working directory name
ranlib	convert archives to random libraries
ratfor	rational Fortran dialect
readnews	read news articles
recnews	receive unprocessed articles via mail

Table of Contents

refer	find and insert literature references in documents
reform	reformat text file
regcmp	regular expression compile
rev	reverse lines of a file
rjestat	RJE status report and interactive status console
rm	remove files or directories
rmdel	remove a delta from an SCCS file
rxctrl	floppy disk manipulating program
sact	print current SCCS file editing activity
sag	system activity graph
sar	system activity reporter
sccsdiff	compare two versions of an SCCS file
script	make typescript of terminal session
sdiff	side-by-side difference program
sed	stream editor
send	gather files and submit RJE jobs
sh	shell, the standard/restricted command programming language
size	size of an object file
sleep	suspend execution for an interval
sno	SNOBOL interpreter
soelim	eliminate .so's from nroff input
sort	sort or merge files
spell	find spelling errors
spline	interpolate smooth curve
split	split a file into pieces
stctrl	special streamer features, skip files
stksiz	set stacksize
strings	find the printable strings in a object, or other binary, file
strip	remove symbols and relocation bits
struct	structure Fortran programs
stty	set the options for a terminal
style	analyze surface characteristics of a document
su	become super-user or another user
sum	sum and count blocks in a file
tabs	set tabs on a terminal
tail	deliver the last part of a file
tar	tape archiver
tbl	format tables for nroff or troff
tc	phototypesetter simulator
tee	pipe fitting
test	condition evaluation command
time	time a command
timex	time a command; report process data and system activity
tk	paginator for the Tektronix 4014
touch	update date last modified of a file
tplot	graphics filters
tr	translate characters
troff	typeset or format text
true	provide truth values
tsort	topological sort
tty	get terminal name

Table of Contents

ul	do underlining
umask	set file-creation mode mask
uname	print name of current UNIX
unset	undo a previous get of an SCCS file
unify	unify relational data base system
uniq	report repeated lines in a file
units	conversion program
uptime	show how long system has been up
users	compact list of users who are on the system
uucp	unix to unix copy
uustat	uucp status inquiry and job control
uuto	public UNIX-to-UNIX file copy
uux	unix to unix command execution
val	validate SCCS file
vc	version control
vfontinfo	inspect and print out information about fonts
vi	screen oriented (visual) display editor based on ex
w	who is on and what they are doing
wait	await completion of process
wc	word count
what	identify SCCS files
whatis	describe what a command is
whereis	locate source, binary, and or manual for program
which	locate a program file including aliases and paths (csh only)
who	who is on the system
write	write to another user
xargs	construct argument list(s) and execute command
xd	hexadecimal dump
xref	cross-reference listing
xstr	extract strings from C programs to implement shared strings
yacc	yet another compiler-compiler
yes	be repetitively affirmative

2. System Calls

intro	introduction to system calls and error numbers
access	determine accessibility of a file
acct	enable or disable process accounting
alarm	set a process's alarm clock
brk	change data segment space allocation
chdir	change working directory
chmod	change mode of file
chown	change owner and group of a file
chroot	change root directory
close	close a file descriptor
creat	create a new file or rewrite an existing one
dup	duplicate an open file descriptor
exec	execute a file
exit	terminate process
fcntl	file control
fork	create a new process
getpid	get process, process group, and parent process IDs

Table of Contents

getuid	get real user, effective user, real group, and effective group IDs
hertz	get the line frequency on the current machine
ioctl	control device
kill	send a signal to a process or a group of processes
link	link to a file
long	system calls modified for long arguments
lseek	move read/write file pointer
mknod	make a directory, or a special or ordinary file
mount	mount a file system
nice	change priority of a process
open	open for reading or writing
pause	suspend process until signal
pipe	create an interprocess channel
plock	lock process, text, or data in memory
profil	execution time profile
ptrace	process trace
read	read from file
setpgrp	set process group ID
setuid	set user and group IDs
signal	specify what to do upon receipt of a signal
stat	get file status
stime	set time
sync	update super-block
time	get time
times	get process and child process times
ulimit	get and set user limits
umask	set and get file creation mask
umount	unmount a file system
uname	get name/ethernet-identification of current UNIX system
unlink	remove directory entry
ustat	get file system statistics
utime	set file access and modification times
wait	wait for child process to stop or terminate
write	write on a file

3. Subroutines

intro	introduction to subroutines and libraries
a64l	convert between long and base-64 ASCII
abort	generate an IOT fault
abort	terminate Fortran program
abs	integer absolute value
abs	Fortran absolute value
acos	Fortran arccosine intrinsic function
aimag	Fortran imaginary part of complex argument
aint	Fortran integer part intrinsic function
asin	Fortran arcsine intrinsic function
assert	verify program assertion
atan	Fortran arctangent intrinsic function
atan2	Fortran arctangent intrinsic function
atof	convert ASCII to numbers
bessel	Bessel functions

Table of Contents

bip	basic functions for BIP, the PCS bitmap display
bool	Fortran bitwise boolean functions
bsearch	binary search
clock	report CPU time used
conjg	Fortran complex conjugate intrinsic function
conv	character translation
cos	Fortran cosine intrinsic function
cosh	Fortran hyperbolic cosine intrinsic function
crypt	DES encryption
ctermid	generate file name for terminal
ctime	convert date and time to ASCII
ctype	classify characters
curses	screen functions with "optimal" cursor motion
cuserid	character login name of the user
d	text database functions
dbm	data base subroutines
drand48	generate uniformly distributed pseudo-random numbers
ecvt	output conversion
end	last locations in program
erf	error function and complementary error function
exp	Fortran exponential intrinsic function
exp	exponential, logarithm, power, square root functions
fclose	close or flush a stream
ferror	stream status inquiries
floor	floor, ceiling, remainder, absolute value functions
fopen	open a stream
fp	Floating Point on the CADMUS
fread	buffered binary input/output
frexp	split into mantissa and exponent
fseek	reposition a stream
ftw	walk a file tree
ftype	explicit Fortran type conversion
gamma	log gamma function
getarg	return Fortran command-line argument
getc	get character or word from stream
getcwd	get path-name of current working directory
getenv	value for environment name
getenv	return Fortran environment variable
getgrent	get group file entry
getlogin	get login name
getopt	get option letter from argv
getpass	read a password
getpw	get name from UID
getpwent	get password file entry
gets	get a string from a stream
getut	access utmp file entry
hsearch	manage hash search tables
hypot	Euclidean distance function
iargc	Number of command-line arguments
index	return location of Fortran substring
l3tol	convert between 3-byte integers and long integers

Table of Contents

len	return length of Fortran string
log	Fortran natural logarithm intrinsic function
log10	Fortran common logarithm intrinsic function
logname	login name of user
long	standard procedures modified for long arguments
lsearch	linear search and update
malloc	main memory allocator
matherr	error-handling function
max	Fortran maximum-value functions
mclock	return Fortran time accounting
min	Fortran minimum-value functions
mktemp	make a unique file name
mod	Fortran remaindering intrinsic functions
monitor	prepare execution profile
nfcomment	a user interface to the notesfile system
nlist	get entries from name list
perror	system error messages
plot	graphics interface subroutines
popen	initiate I/O to/from a process
printf	output formatters
putc	put character or word on a stream
putpwent	write password file entry
puts	put a string on a stream
qsort	quicker sort
rand	random number generator
rand	Fortran uniform random-number generator
regex	regular expression compile/execute
round	Fortran nearest integer functions
scanf	formatted input conversion
setbuf	assign buffering to a stream
setjmp	non-local goto
sign	Fortran transfer-of-sign intrinsic function
signal	specify Fortran action on receipt of a system signal
sin	Fortran sine intrinsic function
sinh	Fortran hyperbolic sine intrinsic function
sinh	hyperbolic functions
sleep	suspend execution for interval
sqrt	Fortran square root intrinsic function
ssignal	software signals
stdio	standard buffered input/output package
string	string operations
swab	swap bytes
system	issue a shell command from Fortran
system	issue a shell command
tan	Fortran tangent intrinsic function
tanh	Fortran hyperbolic tangent intrinsic function
termcap	terminal independent operation routines
tmpfile	create a temporary file
tmpnam	create a name for a temporary file
trig	trigonometric functions
tsearch	manage binary search trees

Table of Contents

ttyname	find name of a terminal
ttyslot	find the slot in the utmp file of the current user
ungetc	push character back into input stream

4. Special Files

intro	introduction to special files
bbp	Basic Block Port Interface
bip	CADMUS Bitmap Display
confinfo	table of interrupt vector and device addresses
dz	DZ-11, DH-11 asynchronous multiplexers
hk	RK611/RK06, RK07 moving-head disk
hp	RP04/05/06, RM02/03 moving-head disk
kl	KL-11 or DL-11 asynchronous interface
lbp	LBP-10 Laser Beam Printer Interface
lp	parallel line printer
mem	core memory
null	the null file
ot	TM 503/TM 603/TM 703 disk, TM 100-4 floppy disk
pipe	Pipes and named pipes
rk	RK-11/RK03 or RK05 disk
rl	RL01/RL02 moving-head disk
rm	RM02/03/05 moving-head disk
rx	RX01 or RX02 floppy disk
sbp	Simplified Basic Block Port Interface
st	SCT11 Streamer interface
termio	general terminal interface
tm	TM-11/TU-10 magtape interface, TS-11 magtape interface
tty	controlling terminal interface
vp	Versatec printer-plotter

5. File Formats

intro	introduction to file formats
a.out	format of programs and modules
acct	per-process accounting file format
ahdr	format of programs and modules
ar	archive (library) file format
checklist	list of file systems processed by fsck
core	format of core image file
cpio	format of cpio archive
dir	format of directories
dump	incremental dump format
ethmap	table of ethernet addresses
fs	file system format of system volume
fspec	format specification in text files
gettydefs	speed and terminal settings used by getty
group	group file
inittab	script for the init process
inode	format of an inode
issue	issue identification file
keycap	keyboard capability data base

Table of Contents

mnttab	mounted file system table
news	USENET network news article, utility files
newsrc	information file for readnews(1) and checknews(1)
passwd	password file
plot	graphics interface
profile	setting up an environment at login time
pwmap	user and group id mappings for the Newcastle Connection
sccsfile	format of SCCS file
stack	C stack frame layout
termcap	terminal capability data base
utab	neighbours known to the Newcastle Connection
utmp	utmp and wtmp entry formats
vfont	font formats for the Benson-Varian or Versatec

6. Games

intro	introduction to games
arithmetic	provide drill in number facts
back	the game of backgammon
backgammon	the game
banner	make long posters
bcd	convert to antique media
bj	the game of black jack
craps	the game of craps
hangman	guess the word
moo	guessing game
quiz	test your knowledge
reversi	reversi, a game of dramatic reversals
rogue	Exploring The Dungeons of Doom
startrek	THE game based on the t.v. series.
ttt	tic-tac-toe
wump	the game of hunt-the-wumpus

7. Miscellaneous Facilities

intro	introduction to miscellany
ascii	map of ASCII character set
environ	user environment
eqnchar	special character definitions for eqn and neqn
fcntl	file control options
greek	graphics for the extended TTY-37 type-box
hier	file system hierarchy
man	macros for formatting entries in this manual
me	macros for formatting papers
mm	the MM macro package for formatting documents
mosd	the OSDD adapter macro package for formatting documents
mptx	the macro package for formatting a permuted index
ms	macros for formatting manuscripts
mv	a macro package for making view graphs
regexp	regular expression compile and match routines
stat	data returned by stat system call
term	conventional names

types primitive system data types

8. System Maintenance Procedures

intro introduction to system maintenance procedures
 NCsetup initialise the Newcastle Connection tables in a process
 acct overview of accounting and miscellaneous accounting commands
 acctcms command summary from per-process accounting records
 acctcon connect-time accounting
 acctmerg merge or add total accounting files
 acctprc process accounting
 acctsh shell procedures for accounting
 bcopy interactive block copy
 boot standalone startup program
 brc system initialization shell scripts
 catman create the cat files for the manual
 check disk checking and formatting
 checkall faster file system checking procedure
 chroot change root directory for a command
 clri clear i-node
 crash crash - what to do when the system crashes
 cron clock daemon
 dcopy copy file systems for optimal access time
 devnm device name
 df report number of free disk blocks
 dmesg collect system diagnostic messages to form error log
 dump incremental file system dump
 dumpdir print the names of files on a dump tape
 expire remove outdated news articles
 ff list file names and statistics for a file system
 filesave daily/weekly UNIX file system backup
 format how to format disks
 fsba file system block analyzer
 fsck file system consistency check and interactive repair
 fsdb file system debugger
 fuser identify processes using a file or file structure
 fwtmp manipulate connect accounting records
 getty set terminal type, modes, speed, and line discipline
 init process control initialization
 install install commands
 killall kill all active processes
 link exercise link and unlink system calls
 lpd line printer daemon
 makekey generate encryption key
 mkalias create an alias to a remote file
 mkfs construct a file system
 mknod build special file
 mksys make a remote system node
 mount mount and dismount file system
 mvdir move a directory
 ncheck generate names from i-numbers
 newconf generate configuration file and reconfigure MUNIX

Table of Contents

notes	notesfile utility programs
pstat	print system facts
pwck	password/group file checkers
quot	summarize file system ownership
recnews	receive unprocessed articles via mail
renice	alter priority of running process by changing nice
restor	incremental file system restore
rje	RJE (Remote Job Entry) to IBM
rmsys	remove a remote system name
runacct	run daily accounting
sa	system accounting
sar	system activity report package
sendnews	send news articles via mail
setmnt	establish mount table
setugi	alter user id of a UNIX server
shutdown	terminate all processing
startnc	starts up (closes down) the file server spawner
sync	update the super block
unite	enable a remote user to access the local system
usam	initiate a UNIX server for a remote client
usrv	UNIX server for a remote client
uuclean	uucp spool directory clean-up
uucp	uucp installation made easy
uurec	receive processed news articles via mail
uusub	monitor uucp network
volcopy	copy file systems with label checking
wall	write to all users
whatconf	what device drivers are in an unix kernel
whodo	who is doing what

PERMUTED INDEX

echo: echo arguments
 hcreate, hdestroy: manage hash search tables
 pack: packs or unpacks many files
 system: issue a shell command
 construct argument list(s) and execute command
 ot, ox: TM 503/TM 603/TM 703 disk, TM
 hp: handle special functions of HP 2840 and
 hp: handle special functions of HP
 l3tol, ltol3: convert between
 dift3: 3-way differential file comparison.
 tk: paginator for the Tektronix
 ot, ox: TM
 ot, ox: TM 503/TM
 ot, ox: TM 503/TM 603/TM
 f77: Fortran
 value.
 utime: set file
 unite: enable a remote user to
 dcopy: copy file systems for optimal
 getutline, pututline, setutent, endutent, utmpname:
 acct: enable or disable process
 acctcon1, acctcon2: connect-time
 acctprc1, acctprc2: process
 shutacct, startup, turnacct: shell procedures for
 acctdisk, acctdusg, accton, acctwtmp: overview of
 acctwtmp: overview of accounting and miscellaneous
 acct: per-process
 acctcom: search and print process
 acctmerg: merge or add total
 mclock: return Fortran time
 acctems: command summary from per-process
 fwtmp, wtmpfix: manipulate connect
 runacct: run daily
 sa, accton: system
 accounting records.
 file(s).
 acctcon1,
 accounting and miscellaneous accounting commands
 and miscellaneous accounting commands. acctdisk,
 miscellaneous accounting/ acctdisk, acctdusg,
 sa,
 acctprc1,
 accounting commands. acctdisk, acctdusg, accton,
 sin, cos, tan, asin,
 killall: kill all
 sag: system
 sa1, sa2, sadc: system
 sar: system
 sact: print current SCCS file editing
 time a command; report process data and system
 mosd: the OSDD
 information: table of interrupt vector and device
 /etc/map_port_cadr: table of ethernet
 yes: be repetitively
 basename: strip filename
 learn: computer
 argument.
 function.
 alarm: set a process's
 /etc/mkalias: create an
 which: locate a program file including
 hsearch,
 <--> one.
 xargs:
 100-4 floppy disk.
 2821-series terminals.
 2840 and 2821-series terminals.
 3-byte integers and long integers.
 3-way differential file comparison.
 4014.
 503/TM 603/TM 703 disk, TM 100-4 floppy disk.
 603/TM 703 disk, TM 100-4 floppy disk.
 703 disk, TM 100-4 floppy disk.
 77 compiler.
 a64l, l64a: convert between long and base-64 ASCII.
 a68: MIT assembler.
 abort: generate an IOT fault.
 abort: terminate Fortran program.
 abs, iabs, dabs, cabs, zabs: Fortran absolute
 abs: integer absolute value.
 access and modification times.
 access: determine accessibility of a file.
 access the local system.
 access time.
 access utmp file entry. getutent, getutid,
 accounting.
 accounting.
 accounting.
 accounting.
 accounting. /prctmp, prdaily, prtacct, runacct,
 accounting and miscellaneous accounting commands.
 accounting commands. acctdisk, acctdusg, accton,
 accounting file format.
 accounting file(s).
 accounting files.
 accounting.
 accounting records.
 accounting records.
 accounting.
 acct: enable or disable process accounting.
 acct: per-process accounting file format.
 acctems: command summary from per-process
 acctcom: search and print process accounting
 acctcon1, acctcon2: connect-time accounting.
 acctcon2: connect-time accounting.
 acctdisk, acctdusg, accton, acctwtmp: overview of
 acctdusg, accton, acctwtmp: overview of accounting
 acctmerg: merge or add total accounting files.
 accton, acctwtmp: overview of accounting and
 accton: system accounting.
 acctprc1, acctprc2: process accounting.
 acctprc2: process accounting.
 acctwtmp: overview of accounting and miscellaneous
 acos, atan, atan2: trigonometric functions.
 acos, dacos: Fortran arccosine intrinsic function.
 active, processes.
 activity graph.
 activity report package.
 activity reporter.
 activity.
 activity. timex:
 adapter macro package for formatting documents.
 adb: debugger.
 addresses. configuration
 addresses.
 admin: create and administer SCCS files.
 affixes.
 aided instruction about UNIX.
 aimag, dimag: Fortran imaginary part of complex
 aint, dint: Fortran integer part intrinsic
 alarm clock.
 alarm: set a process's alarm clock.
 alias to a remote file.
 aliases and paths (csh only).
 echo(1)
 hsearch(3C)
 pack(1)
 system(3S)
 xargs(1)
 ot(4)
 hp(1)
 hp(1)
 l3tol(3C)
 dift3(1)
 tk(1)
 ot(4)
 ot(4)
 ot(4)
 f77(1)
 a64l(3C)
 a68(1)
 abort(3C)
 abort(3f)
 abs(3f)
 abs(3C)
 utime(2)
 access(2)
 unite(8N)
 dcopy(8)
 getut(3C)
 acct(2)
 acctcon(8)
 acctprc(8)
 acctsh(8)
 acct(8)
 acct(8)
 acct(5)
 acctcom(1)
 acctmerg(8)
 mclock(3f)
 acctems(8)
 fwtmp(8)
 runacct(8)
 sa(8)
 acct(2)
 acct(5)
 acctems(8)
 acctcom(1)
 acctcon(8)
 acct(8)
 acct(8)
 acctmerg(8)
 sa(8)
 acctprc(8)
 acctprc(8)
 acct(8)
 trig(3M)
 acos(3f)
 killall(8)
 sag(1G)
 sar(8)
 sar(1)
 sact(1)
 timex(1)
 mosd(7)
 adb(1)
 confinfo(4)
 ethmap(5N)
 admin(1)
 yes(1)
 basename(1)
 learn(1)
 aimag(3f)
 aint(3f)
 alarm(2)
 alarm(2)
 mkalias(8N)
 which(1)

Permuted Index

brk, sbrk: change data segment space	allocation.	brk(2)
malloc, free, realloc, calloc: main memory	allocator.	malloc(3C)
intrinsic function. log,	alog, dlog, clog: Fortran natural logarithm	log(3f)
function. log10,	alog10, dlog10: Fortran common logarithm intrinsic	log10(3f)
renice:	alter priority of running process by changing nice.	renice(8)
/etc/setugi:	alter user id of a UNIX server.	setugi(8N)
functions. max, max0,	amax0, max1, amax1, dmax1: Fortran maximum-value	max(3f)
max, max0, amax0, max1,	amax1, dmax1: Fortran maximum-value functions.	max(3f)
functions. min, min0,	amin0, min1, amin1, dmin1: Fortran minimum-value	min(3f)
min, min0, amin0, min1,	amin1, dmin1: Fortran minimum-value functions.	min(3f)
functions. mod,	amod, dmod: Fortran remaindering intrinsic	mod(3f)
lex: generator of lexical	analysis programs.	lex(1)
error:	analyze and disperse compiler error messages.	error(1)
style:	analyze surface characteristics of a document.	style(1)
fsba: file system block	analyzer.	fsba(8)
boolean functions.	and, or, xor, not, lshift, rshift: Fortran bitwise	bool(3f)
functions.	anint, dnint, mint, idmint: Fortran nearest integer	round(3f)
bcd, ppt: convert to	antique media.	bcd(8)
	a.out: format of programs and modules.	ahdr(5)
	a.out: format of programs and modules.	a.out(5)
	apropos: locate commands by keyword lookup.	apropos(1)
	ar: archive and library maintainer.	ar(1)
	ar: archive (library) file format.	ar(5)
	arbitrary-precision arithmetic language.	bc(1)
bc:	arccosine intrinsic function.	acos(3f)
acos, dacos: Fortran	archive and library maintainer.	ar(1)
ar:	archive.	cpio(5)
cpio: format of cpio	archive (library) file format.	ar(5)
ar:	archiver.	tar(1)
tar: tape	archives in and out.	cpio(1)
cpio: copy file	archives to random libraries.	ranlib(1)
ranlib: convert	arcsine intrinsic function.	asin(3f)
asin, dasin: Fortran	arctangent intrinsic function.	atan2(3f)
atan2, datan2: Fortran	arctangent intrinsic function.	atan(3f)
atan, datan: Fortran	argument list(s) and execute command.	xargs(1)
xargs: construct	arguments.	iargc(3f)
iargc: Number of command-line	arguments.	long(3)
long: system calls modified for long	arguments.	long(3C)
long: standard procedures modified for long	argv.	getopt(3C)
getopt: get option letter from	arithmetic language.	bc(1)
bc: arbitrary-precision	arithmetic: provide drill in number facts.	arithmetic(6)
	article, utility files.	news(5)
news: USENET network news	as an expression.	expr(1)
expr: evaluate arguments	as: assembler.	as(1)
	ASCII.	a64l(3C)
a64l, l64a: convert between long and base-64	ASCII character set.	ascii(7)
ascii: map of	ASCII. ctime, localtime,	ctime(3C)
gmtime, asctime, tzset: convert date and time to	ascii: map of ASCII character set.	ascii(7)
	ASCII to numbers.	atoi(3C)
atof, atoi, atol: convert	asctime, tzset: convert date and time to ASCII.	ctime(3C)
ctime, localtime, gmtime,	asin, acos, atan, atan2: trigonometric functions.	trig(3W)
sin, cos, tan,	asin, dasin: Fortran arcsine intrinsic function.	asin(3f)
	ask for help.	help(1)
help:	assembler.	a68(1)
a68: M68	assembler.	as(1)
as:	assert: verify program assertion.	assert(3X)
assert: verify program	assertion.	assert(3X)
assert:	assign buffering to a stream.	setbuf(3S)
setbuf:	asynchronous interface.	kl(4)
kl: KL-11 or DL-11	asynchronous multiplexers.	dz(4)
dz, dh: DZ-11, DH-11	atan, atan2: trigonometric functions.	trig(3W)
sin, cos, tan, asin, acos,	atan, datan: Fortran arctangent intrinsic function.	atan(3f)
	atan2, datan2: Fortran arctangent intrinsic	atan2(3f)
function.	atan2: trigonometric functions.	trig(3W)
sin, cos, tan, asin, acos, atan,	atof, atoi, atol: convert ASCII to numbers.	atoi(3C)
	atoi, atoi: convert ASCII to numbers.	atoi(3C)
atof, atoi,	atol: convert ASCII to numbers.	atol(3C)
notes,	autoeq, nfpip, nfpint, nfstats: a news system.	notes(1)
wait:	await completion of process.	wait(1)
	awk: pattern scanning and processing language.	awk(1)
ungetc: push character	back into input stream.	ungetc(3S)
	back: the game of backgammon.	back(8)
back: the game of	backgammon.	back(8)
	backgammon: the game.	backgammon(8)
filesave, tapesave: daily/weekly UNIX file system	backup.	filesave(8)
	banner: make long posters.	banner(8)
a64l, l64a: convert between long and	base-64 ASCII.	a64l(3C)
vi: screen oriented (visual) display editor	based on ex.	vi(1)
startrek: THE game	based on the t.v. series..	startrek(8)

Permuted Index

basename: strip filename affixes.	basename(1)
basic: Basic Interpreter.	basic(1)
bbp: Basic Block Port Interface.	bbp(4)
sbp: Simplified Basic Block Port Interface.	sbp(4)
Bip: basic functions for BiP, the PCS bitmap display.	bip(3)
basic: Basic Interpreter.	basic(1)
bbp: Basic Block Port Interface.	bbp(4)
bc: arbitrary-precision arithmetic language.	bc(1)
bcd, ppt: convert to antique media.	bcd(8)
bcheckrc, rc, powerfail: system initialization	brc(8)
bcopy: interactive block copy.	bcopy(8)
bdiff: big diff.	bdiff(1)
Beam Printer Interface.	lbp(4)
beautifier.	cb(1)
Benson-Varian or Versatec.	vfont(5)
Bessel functions.	bessel(3M)
bfs: big file scanner.	bfs(1)
binary, and or manual for program.	whereis(1)
binary, file, strings:	strings(1)
binary input/output.	fread(3S)
binary search.	bsearch(3C)
binary search trees.	tsearch(3C)
Bip: basic functions for BiP, the PCS bitmap	bip(3)
bip: CADMUS Bitmap Display.	bip(4)
BiP, the PCS bitmap display.	bip(3)
bitmap display.	bip(3)
Bitmap Display.	bip(4)
bitwise boolean functions.	bool(3f)
bj: the game of black jack.	bj(8)
black jack.	bj(8)
block analyzer.	fsba(8)
block copy.	bcopy(8)
Block Port Interface.	bbp(4)
Block Port Interface.	sbp(4)
block.	sync(8)
blocks.	df(1)
blocks.	df(8)
boolean functions.	bool(3f)
boot: standalone startup program.	boot(8)
brc, bcheckrc, rc, powerfail: system initialization	brc(8)
brk, sbrk: change data segment space allocation.	brk(2)
bs: a compiler/interpreter for modest-sized	bs(1)
bsearch: binary search.	bsearch(3C)
buffered binary input/output.	fread(3S)
buffered input/output package.	stdio(3S)
buffering to a stream.	setbuf(3S)
build special file.	mknod(8)
bytes.	swab(3C)
C compiler.	cc(1)
C language preprocessor.	cpre(1)
C program beautifier.	cb(1)
C program source.	indent(1)
C program verifier.	lint(1)
C programs to implement shared strings.	xstr(1)
C source.	mkstr(1)
C stack frame layout.	stack(5)
cabs, zabs: Fortran absolute value.	abs(3f)
CADMUS.	fp(3)
cal: print calendar.	cal(1)
calculator.	dc(1)
calendar.	cal(1)
calendar: reminder service.	calendar(1)
call.	stat(7)
call UNIX.	cu(1C)
calloc: main memory allocator.	malloc(3C)
calls and error numbers.	intro(2)
calls.	link(8)
calls modified for long arguments.	long(2)
CANON LBP or the Versatec V80.	ltroff(1)
capability data base.	keycap(5)
capability data base.	termcap(5)
cat: concatenate and print files.	cat(1)
cat files for the manual.	catman(8)
cat them, compact.	compact(1)
catman: create the cat files for the manual.	catman(8)
cb: C program beautifier.	cb(1)
cc, lcc: C compiler.	cc(1)
ccat: compress and uncompress files, and cat them.	compact(1)
ccos: Fortran cosine intrinsic function.	cos(3f)
bbp: Basic Block Port Interface.	bbp(4)
sbp: Simplified Basic Block Port Interface.	sbp(4)
Bip: basic functions for BiP, the PCS bitmap display.	bip(3)
basic: Basic Interpreter.	basic(1)
bbp: Basic Block Port Interface.	bbp(4)
bc: arbitrary-precision arithmetic language.	bc(1)
bcd, ppt: convert to antique media.	bcd(8)
bcheckrc, rc, powerfail: system initialization	brc(8)
bcopy: interactive block copy.	bcopy(8)
bdiff: big diff.	bdiff(1)
Beam Printer Interface.	lbp(4)
beautifier.	cb(1)
Benson-Varian or Versatec.	vfont(5)
Bessel functions.	bessel(3M)
bfs: big file scanner.	bfs(1)
binary, and or manual for program.	whereis(1)
binary, file, strings:	strings(1)
binary input/output.	fread(3S)
binary search.	bsearch(3C)
binary search trees.	tsearch(3C)
Bip: basic functions for BiP, the PCS bitmap	bip(3)
bip: CADMUS Bitmap Display.	bip(4)
BiP, the PCS bitmap display.	bip(3)
bitmap display.	bip(3)
Bitmap Display.	bip(4)
bitwise boolean functions.	bool(3f)
bj: the game of black jack.	bj(8)
black jack.	bj(8)
block analyzer.	fsba(8)
block copy.	bcopy(8)
Block Port Interface.	bbp(4)
Block Port Interface.	sbp(4)
block.	sync(8)
blocks.	df(1)
blocks.	df(8)
boolean functions.	bool(3f)
boot: standalone startup program.	boot(8)
brc, bcheckrc, rc, powerfail: system initialization	brc(8)
brk, sbrk: change data segment space allocation.	brk(2)
bs: a compiler/interpreter for modest-sized	bs(1)
bsearch: binary search.	bsearch(3C)
buffered binary input/output.	fread(3S)
buffered input/output package.	stdio(3S)
buffering to a stream.	setbuf(3S)
build special file.	mknod(8)
bytes.	swab(3C)
C compiler.	cc(1)
C language preprocessor.	cpre(1)
C program beautifier.	cb(1)
C program source.	indent(1)
C program verifier.	lint(1)
C programs to implement shared strings.	xstr(1)
C source.	mkstr(1)
C stack frame layout.	stack(5)
cabs, zabs: Fortran absolute value.	abs(3f)
CADMUS.	fp(3)
cal: print calendar.	cal(1)
calculator.	dc(1)
calendar.	cal(1)
calendar: reminder service.	calendar(1)
call.	stat(7)
call UNIX.	cu(1C)
calloc: main memory allocator.	malloc(3C)
calls and error numbers.	intro(2)
calls.	link(8)
calls modified for long arguments.	long(2)
CANON LBP or the Versatec V80.	ltroff(1)
capability data base.	keycap(5)
capability data base.	termcap(5)
cat: concatenate and print files.	cat(1)
cat files for the manual.	catman(8)
cat them, compact.	compact(1)
catman: create the cat files for the manual.	catman(8)
cb: C program beautifier.	cb(1)
cc, lcc: C compiler.	cc(1)
ccat: compress and uncompress files, and cat them.	compact(1)
ccos: Fortran cosine intrinsic function.	cos(3f)
whereis: locate source, find the printable strings in a object, or other	
fread, fwrite: buffered	
bsearch:	
tsearch, tdelete, twalk: manage display.	
Bip: basic functions for	
Bip: basic functions for BiP, the PCS	
bip: CADMUS	
and, or, xor, not, lshift, rshift: Fortran	
bj: the game of	
fsba: file system	
bcopy: interactive	
bbp: Basic	
sbp: Simplified Basic	
sync: update the super	
df: report number of free disk	
df: report number of free disk	
and, or, xor, not, lshift, rshift: Fortran bitwise	
shell scripts.	
programs.	
fread, fwrite:	
stdio: standard	
setbuf: assign	
mknod:	
swab: swap	
cc, lcc:	
cpre: the	
cb:	
indent: indent and format a	
lint: a	
xstr: extract strings from	
mkstr: create an error message file by massaging	
abs, labs, dabs,	
fp: Floating Point on the	
dc: desk	
cal: print	
stat: data returned by stat system	
cu:	
malloc, free, realloc,	
intro: introduction to system	
link, unlink: exercise link and unlink system	
long: system	
ltroff, vtroff: troff to the	
keycap: keyboard	
termcap: terminal	
catman: create the	
uncompact, ccat: compress and uncompress files, and	
compact, uncompact,	
cos, dcos,	

February 24, 1984

- 4 -

Permuted Index

regcmp: regular expression	compile.	regcmp(1)
regex, regcmp: regular expression	compile/execute.	regex(3X)
cc, lcc: C	compiler.	cc(1)
error: analyze and disperse	compiler error messages.	error(1)
f77: Fortran 77	compiler.	f77(1)
pc: Pascal	compiler.	pc(1)
yacc: yet another	compiler-compiler.	yacc(1)
bs: a	compiler/interpreter for modest-sized programs.	bs(1)
erf, erfc: error function and	complementary error function.	erf(3M)
wait: await	completion of process.	wait(1)
aimag, dimag: Fortran imaginary part of	complex argument.	aimag(3f)
conjg, dconjg: Fortran	complex conjugate intrinsic function.	conjg(3f)
compact, uncompact, ccat:	compress and uncompress files, and cat them.	compact(1)
learn:	computer aided instruction about UNIX.	learn(1)
cat:	concatenate and print files.	cat(1)
test:	condition evaluation command.	test(1)
newconf: generate	configuration file and reconfigure MUNIX.	newconf(8)
vector and device addresses.	configuration information: table of interrupt	confinfo(4)
function.	conjg, dconjg: Fortran complex conjugate intrinsic	conjg(3f)
conjg, dconjg: Fortran complex	conjugate intrinsic function.	conjg(3f)
ftmp, wtmpfix: manipulate	connect accounting records.	ftmp(8)
user and group id mappings for the Newcastle	Connection. /etc/pwmap, /etc/groupmap:	pwmap(5N)
/etc/utab: neighbours known to the Newcastle	Connection.	utab(5N)
acctcon1, acctcon2:	connect-time accounting.	acctcon(8)
fsck, dfck: file system	consistency check and interactive repair.	fsck(8)
rjstat: RJE status report and interactive status	console.	rjstat(1C)
mkfs:	construct a file system.	mkfs(8)
xargs:	construct argument list(s) and execute command	xargs(1)
deroff: remove nroff/troff, tbl, and eqn	constructs.	deroff(1)
ls, ll, l, lr, lx: list	contents of directory.	ls(1)
csplit:	context split.	csplit(1)
ioctl:	control device.	ioctl(2)
fentl: file	control.	fentl(2)
init, telinit: process	control initialization.	init(8)
fentl: file	control options.	fentl(7)
uustat: uucp status inquiry and job	control.	uustat(1C)
vc: version	control.	vc(1)
tty:	controlling terminal interface.	tty(4)
term:	conventional names.	term(7)
ecvt, fcvt: output	conversion.	ecvt(3C)
cmplx, demplx, ichar, char: explicit Fortran type	conversion. /ifx, idint, real, float, singl, dble.	ftype(3f)
units:	conversion program.	units(1)
scanf, fscanf, sscanf: formatted input	conversion.	scanf(3S)
dd:	convert and copy a file.	dd(1)
ranlib:	convert archives to random libraries.	ranlib(1)
atof, atoi, atol:	convert ASCII to numbers.	atof(3C)
l3tol, ltol3:	convert between 3-byte integers and long integers.	l3tol(3C)
a64l, l64a:	convert between long and base-64 ASCII	a64l(3C)
ctime, localtime, gmtime, asctime, tzset:	convert date and time to ASCII.	ctime(3C)
bcd, ppt:	convert to antique media.	bcd(8)
dd: convert and	copy a file.	dd(1)
bcopy: interactive block	copy.	bcopy(8)
cpio:	copy file archives in and out.	cpio(1)
dcopy:	copy file systems for optimal access time.	dcopy(8)
volcopy, labelit:	copy file systems with label checking.	volcopy(8)
cp, ln, mv:	copy, link or move files.	cp(1)
uucp, uulog, uuname: unix to unix	copy.	uucp(1C)
uuto, uupick: public UNIX-to-UNIX file	copy.	uuto(1C)
	core: format of core image file.	core(5)
	core image file.	core(5)
core: format of	core memory.	mem(4)
mem, kmem:	cos, dcosh, ccosh: Fortran cosine intrinsic function.	cos(3f)
	cos, tan, asin, acos, atan, atan2: trigonometric	trig(3M)
functions, sin,	cosh, dcosh: Fortran hyperbolic cosine intrinsic	cosh(3f)
function.	cosh, tanh: hyperbolic functions.	sinh(3M)
sinh,	cosine intrinsic function.	cos(3f)
cos, dcosh, ccosh: Fortran	cosine intrinsic function.	cosh(3f)
cosh, dcosh: Fortran hyperbolic	count blocks in a file.	sum(1)
sum: sum and	count.	wc(1)
wc: word	cp, ln, mv: copy, link or move files.	cp(1)
	cpio archive.	cpio(5)
cpio: format of	cpio: copy file archives in and out.	cpio(1)
	cpio: format of cpio archive.	cpio(5)
	cpp: the C language preprocessor.	cpp(1)
clock: report	CPU time used.	clock(3C)
craps: the game of	craps.	craps(8)
	craps: the game of craps.	craps(8)
	crash - what to do when the system crashes.	crash(8)
crash - what to do when the system	crashes.	crash(8)

Permuted Index

one.	creat: create a new file or rewrite an existing	creat(2)
umask: set and get file	creation mask.	umask(2)
	cref: make cross-reference listing.	cref(1)
	cron: clock daemon.	cron(8)
	cross-reference listing.	cref(1)
cref: make	cross-reference listing.	xref(1)
xref:	CRT previewing.	colcrt(1)
colcrt: filter nroff output for	crt viewing.	more(1)
more, page: file perusal filter for	crypt: encode/decode.	crypt(1)
	crypt, setkey, encrypt: DES encryption.	crypt(3C)
	csh: a shell (command interpreter) with C-like	csh(1)
	(csh only). which:	which(1)
locate a program file including aliases and paths	csin: Fortran sine intrinsic function.	sin(3f)
sin, dsin,	csplit: context split.	csplit(1)
	csqrt: Fortran square root intrinsic function.	sqrt(3f)
sqrt, dsqrt,	ctags: create a tags file.	ctags(1)
	ctermid: generate file name for terminal.	ctermid(3S)
	ctime, localtime, gmtime, asctime, tzset: convert	ctime(3C)
date and time to ASCII.	cu: call UNIX.	cu(1C)
	current machine.	hertz(2)
hertz: get the line frequency on the	current SCCS file editing activity.	sact(1)
sact: print	current UNIX system.	uname(2)
uname, ethname: get name/ethernet-identification of	current UNIX.	uname(1)
uname: print name of	current user.	ttyslot(3C)
ttyslot: find the slot in the utmp file of the	current working directory.	getcwd(3C)
getcwd: get path-name of	curses: screen functions with "optimal" cursor	curses(3)
motion.	cursor motion.	curses(3)
curses: screen functions with "optimal"	curve.	spline(1G)
spline: interpolate smooth	cuserid: character login name of the user.	cuserid(3S)
	cut: cut out selected fields of each line of a	cut(1)
file.	cut: cut out selected fields of each line of a file.	cut(1)
cut:	d: text database functions.	d(3)
	d: text database functions.	d(1)
	dabs, cabs, zabs: Fortran absolute value.	abs(3f)
abs, iabs,	dacos: Fortran arccosine intrinsic function.	acos(3f)
acos,	daemon.	cron(8)
cron: clock	daemon.	lpd(8)
lpd: line printer	daily accounting.	runacct(8)
runacct: run	daily/weekly UNIX file system backup.	filesave(8)
filesave, tapesave:	dasin: Fortran arcsine intrinsic function.	asin(3f)
asin,	data and system activity.	time(1)
time(1): time a command; report process	data base.	dbadd(1)
dbadd: add entry to an Emacs	data base.	keycap(5)
keycap: keyboard capability	data base subroutines.	dbm(3X)
dbm(1), fetch, store, delete, firstkey, nextkey:	data base system.	unify(1)
unify relational	data base.	termcap(5)
termcap: terminal capability	data in memory.	plock(2)
plock: lock process, text, or	data.	prof(1)
prof: display profile	data returned by stat system call.	stat(7)
stat:	data segment space allocation.	brk(2)
brk, sbrk: change	data types.	types(7)
types: primitive system	database functions.	d(1)
d: text	database functions.	d(3)
d: text	database operator.	join(1)
join: relational	atan: Fortran arctangent intrinsic function.	atan(3f)
atan,	atan2: Fortran arctangent intrinsic function.	atan2(3f)
atan2,	date and time to ASCII.	ctime(3C)
ctime, localtime, gmtime, asctime, tzset: convert	date.	date(1)
date: print and set the	date last modified of a file.	touch(1)
touch: update	date: print and set the date.	date(1)
	dbadd: add entry to an Emacs data base.	dbadd(1)
	dbm, cmplx, demplx, ichar, char: explicit Fortran	fctype(3f)
type/ int, iflx, idint, real, float, snl,	dbm(1), fetch, store, delete, firstkey, nextkey:	dbm(3X)
data base subroutines.	dc: desk calculator.	dc(1)
	demplx, ichar, char: explicit Fortran type/	fctype(3f)
int, iflx, idint, real, float, snl, dbm, cmplx,	dconjg: Fortran complex conjugate intrinsic	conjg(3f)
function. conjg,	dcopy: copy file systems for optimal access time.	dcopy(8)
	dcos, ccos: Fortran cosine intrinsic function.	cos(3f)
cos,	dcosh: Fortran hyperbolic cosine intrinsic	cosh(3f)
function. cosh,	dd: convert and copy a file.	dd(1)
	ddate: incremental dump format.	dump(5)
dump,	debugger.	adb(1)
adb:	debugger.	fsdb(8)
fsdb: file system	definitions for eqn and neqn.	eqnchar(7)
eqnchar: special character	delete, firstkey, nextkey: data base subroutines.	dbm(3X)
dbm(1), fetch, store,	delimiters.	mmchek(1)
mmchek: check usage of mm macros and eqn	deliver the last part of a file.	tail(1)
tail:	delta.	cdc(1)
cdc: change the delta commentary of an SCCS	delta (change) to an SCCS file.	delta(1)
delta: make a		

Permuted Index

cdc: change the	delta commentary of an SCCS delta.	cdc(1)
rm: remove a	delta from an SCCS file.	rm(1)
comb: combine SCCS	delta: make a delta (change) to an SCCS file.	delta(1)
mesg: permit or	deltas.	comb(1)
constructs.	deny messages.	mesg(1)
crypt, setkey, encrypt:	deroff: remove nroff/troff, tbl, and eqn	deroff(1)
what: whatis:	DES encryption.	crypt(3C)
close: close a file	describe what a command is.	whatis(1)
dup: duplicate an open file	descriptor.	close(2)
dc: desk calculator.	descriptor.	dup(2)
access: determine accessibility of a file.	dc: desk calculator.	dc(1)
file: determine file type.	access: determine accessibility of a file.	access(2)
information: table of interrupt vector and	file: determine file type.	file(1)
whatconf: what	device addresses. configuration	confinfo(4)
fold: fold long lines for finite width output	device drivers are in an unix kernel.	whatconf(8)
ioctl: control	device.	fold(1)
devnm: device name.	device.	ioctl(2)
devnm: device name.	device name.	devnm(8)
exp, dexp, cexp: Fortran exponential intrinsic function.	devnm: device name.	devnm(8)
df: report number of free disk blocks.	exp, dexp, cexp: Fortran exponential intrinsic function.	exp(3f)
df: report number of free disk blocks.	df: report number of free disk blocks.	df(1)
fsck: file system consistency check and	df: report number of free disk blocks.	df(8)
dh: DZ-11, DH-11 asynchronous multiplexers.	fsck: file system consistency check and	fsck(8)
DH-11 asynchronous multiplexers.	dh: DZ-11, DH-11 asynchronous multiplexers.	dz(4)
diagnostic messages to form error log.	DH-11 asynchronous multiplexers.	dz(4)
dialect.	diagnostic messages to form error log.	dmesg(8)
diction, diction, explain:	dialect.	ratfor(1)
diction, explain: print wordy sentences.	diction, diction, explain:	diction(1)
diff.	diction, explain: print wordy sentences.	diction(1)
diff: differential file comparator.	diff.	bdiff(1)
diff3: 3-way differential file comparison.	diff: differential file comparator.	diff(1)
difference program.	diff3: 3-way differential file comparison.	diff3(1)
differential file comparator.	difference program.	sdiff(1)
differential file comparison.	differential file comparator.	diff(1)
dimag: Fortran imaginary part of complex argument.	differential file comparison.	diff3(1)
dint: Fortran integer part intrinsic function.	dimag: Fortran imaginary part of complex argument.	aimag(3f)
dir: format of directories.	dint: Fortran integer part intrinsic function.	aint(3f)
dir: format of	dir: format of directories.	dir(5)
directories.	dir: format of	dir(5)
directories.	directories.	dircmp(1)
directory.	directories.	dir(5)
directory.	directory.	rm(1)
directory.	directory.	cd(1)
directory.	directory.	chdir(2)
directory.	directory.	chroot(2)
directory clean-up.	directory.	uuclean(8)
directory comparison.	directory clean-up.	uuclean(8)
directory entry.	directory comparison.	dircmp(1)
directory for a command.	directory entry.	unlink(2)
directory.	directory for a command.	chroot(8)
directory.	directory.	getcwd(3C)
directory.	directory.	ls(1)
directory name.	directory.	mkdir(1)
directory, or a special or ordinary file.	directory name.	mmdir(8)
disable process accounting.	directory, or a special or ordinary file.	pwd(1)
discipline.	disable process accounting.	mknod(2)
disk blocks.	discipline.	acct(2)
disk blocks.	disk blocks.	getty(8)
disk checking and formatting.	disk blocks.	df(1)
disk.	disk checking and formatting.	df(8)
disk.	disk.	check(8)
disk manipulating program.	disk.	hk(4)
disk.	disk manipulating program.	hp(4)
disk.	disk.	rxctrl(1)
disk.	disk.	ot(4)
disk.	disk.	rk(4)
disk.	disk.	rl(4)
disk.	disk.	rm(4)
disk.	disk.	rx(4)
disk, TM 100-4 floppy disk.	disk.	ot(4)
disk usage.	disk, TM 100-4 floppy disk.	du(1)
disks.	disk usage.	format(8)
dismount file system.	disks.	mount(8)
disperse compiler error messages.	dismount file system.	error(1)
display.	disperse compiler error messages.	bip(3)
Display.	display.	bip(4)
display editor based on ex.	Display.	vi(1)
display profile data.	display editor based on ex.	prof(1)
distance function.	display profile data.	hypot(3M)
distributed pseudo-random numbers. /jrand48,	distance function.	drand48(3C)
DL-11 asynchronous interface.	distributed pseudo-random numbers. /jrand48,	kl(4)
	DL-11 asynchronous interface.	

Permuted Index

function. log, alog, dlog, clog: Fortran natural logarithm intrinsic log(3f)
 function. log10, alog10, dlog10: Fortran common logarithm intrinsic log10(3f)
 max, max0, amax0, max1, amax1, dmax1: Fortran maximum-value functions. . . . max(3f)
 error log. dmesg: collect system diagnostic messages to form dmesg(8)
 min, min0, amin0, min1, amin1, dmin1: Fortran minimum-value functions. . . . min(3f)
 mod, amod, dmod: Fortran remaindering intrinsic functions. . . . mod(3f)
 functions. anint, dnint, nint, idnint: Fortran nearest integer round(3f)
 style: analyze surface characteristics of a document. . . . style(1)
 mm: print out documents formatted with the MM macros. . . . mm(1)
 mm: the MM macro package for formatting documents. . . . mm(7)
 mosd: the OSDD adapter macro package for formatting documents. . . . mosd(7)
 lookbib: find and insert literature references in documents. refer. . . . refer(1)
 mmt, mvt: typeset documents, view graphs, and slides. . . . mmt(1)
 prdaily, prtacct, runacct, / chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, w(1)
 w: who is on and what they are whodo: who is doing what. . . . whodo(8)
 rogue: Exploring The Dungeons of Doom. . . . rogue(8)
 /etc/startnc, /etc/stopnc: starts up (closes down) the file server spawner. . . . startnc(8N)
 reversi: reversi, a game of dramatic reversals. . . . reversi(8)
 rand48, srand48, seed48, lcong48: generate / drand48, erand48, lrand48, nrand48, mrand48. . . . drand48(3C)
 graph: draw a graph. . . . graph(1C)
 arithmetic: provide arithmetic(8)
 whatconf: what device drivers are in an unix kernel. . . . whatconf(8)
 sign, isign, dsign: Fortran transfer-of-sign intrinsic function. . . . sign(3f)
 sin, dsin, csin: Fortran sine intrinsic function. . . . sin(3f)
 sinh, dsinh: Fortran hyperbolic sine intrinsic function. . . . sinh(3f)
 function. sqrt, dsqrt, csqrt: Fortran square root intrinsic sqrt(3f)
 tan, dtan: Fortran tangent intrinsic function. . . . tan(3f)
 function. tanh, dtanh: Fortran hyperbolic tangent intrinsic tanh(3f)
 du: summarize disk usage. . . . du(1)
 dump, ddate: incremental dump format. . . . dump(5)
 dump. . . . dump(8)
 dump format. . . . dump(5)
 dump: incremental file system dump. . . . dump(8)
 od: octal dump. . . . od(1)
 dumpdir: print the names of files on a dump tape. . . . dumpdir(8)
 xd: hexadecimal dump. . . . xd(1)
 dumpdir: print the names of files on a dump tape. . . . dumpdir(8)
 Dungeons of Doom. . . . rogue(8)
 dup: duplicate an open file descriptor. . . . dup(2)
 duplicate an open file descriptor. . . . dup(2)
 dz, dh: DZ-11, DH-11 asynchronous multiplexers. . . . dz(4)
 DZ-11, DH-11 asynchronous multiplexers. . . . dz(4)
 easy. . . . uucp(8)
 uucp: uucp installation made echo arguments. . . . echo(1)
 echo: echo arguments. . . . echo(1)
 ecvt, fcvt: output conversion. . . . ecvt(3C)
 ed, red: text editor. . . . ed(1)
 edata: last locations in program. . . . ed(3C)
 editing activity. . . . sact(1)
 editor based on ex. . . . vi(1)
 editor. . . . ed(1)
 editor. . . . emacs(1)
 editor. . . . ex(1)
 editor. . . . med(1)
 editor. . . . sed(1)
 effective group IDs. /geteuid, getgid, getegid: geteuid(2)
 effective user, real group, and effective group / getuid(2)
 efi: Extended Fortran Language. . . . efi(1)
 egrep, fgrep: search a file for a pattern. . . . grep(1)
 eliminate .so's from nroff input. . . . soelim(1)
 emacs: a screen editor. . . . emacs(1)
 Emacs data base. . . . dbadd(1)
 dbadd: add entry to an enable a remote user to access the local system. . . . unite(8N)
 unite: enable or disable process accounting. . . . acct(2)
 acct: encode/decode. . . . crypt(1)
 crypt, setkey, encrypt: DES encryption. . . . crypt(3C)
 crypt, setkey, encrypt: DES encryption. . . . crypt(3C)
 makekey: generate encryption key. . . . makekey(8)
 end, etext, edata: last locations in program. . . . end(3C)
 endgrent: get group file entry. . . . getgrent(3C)
 endpwent: get password file entry. . . . getpwent(3C)
 endutent, utmpname: access utmp file entry. . . . getut(3C)
 entries from name list. . . . nlist(3C)
 entries in this manual. . . . man(1)
 Entry to IBM. . . . man(7)
 env: set environment for command execution. . . . rje(8)
 environ: user environment. . . . env(1)
 environ(7)

Permuted Index

profile: setting up an environment.	environment at login time.	profile(5)
environ: user environment.	environment.	environ(7)
env: set environment for command execution.	environment for command execution.	env(1)
getenv: value for environment name.	environment name.	getenv(3C)
getenv: return Fortran environment variable.	environment variable.	getenv(3F)
eqnchar: special character definitions for eqn and neqn.	eqn and neqn.	eqnchar(7)
deroff: remove nroff/troff, tbl, and mmchek: check usage of mm macros and nroff or troff.	eqn constructs.	deroff(1)
neqn.	eqn delimiters.	mmchek(1)
eqn, neqn, checkeq: format mathematical text for eqnchar: special character definitions for eqn and erand48, lrand48, nrand48, mrand48, jrand48, erf, erfci: error function and complementary error erfci: error function and complementary error	errno: system error messages.	error(1)
error: analyze and disperse compiler error error function and complementary error function.	error function.	error(3M)
error log.	error message file by massaging C source.	error(3M)
error messages.	error messages.	error(1)
error messages.	error numbers.	error(3C)
error-handling function.	errors.	error(1)
errors.	establish mount table.	error(3M)
establish mount table.	/etc/groupmap: user and group id mappings for the	error(1)
/etc/groupmap: user and group id mappings for the	/etc/map_portadr: table of ethernet addresses.	error(3C)
/etc/map_portadr: table of ethernet addresses.	/etc/mkalias: create an alias to a remote file.	error(2)
/etc/mkalias: create an alias to a remote file.	/etc/mksys: make a remote system node.	error(3M)
/etc/mksys: make a remote system node.	/etc/NCsetup: initialise the Newcastle Connection	error(3M)
/etc/NCsetup: initialise the Newcastle Connection	/etc/pwmap, /etc/groupmap: user and group id	error(1)
/etc/pwmap, /etc/groupmap: user and group id	/etc/rmsys: remove a remote system name.	error(3C)
/etc/rmsys: remove a remote system name.	/etc/setugi: alter user id of a UNIX server.	error(1)
/etc/setugi: alter user id of a UNIX server.	/etc/startnc, /etc/stopnc: starts up (closes down)	error(3C)
/etc/startnc, /etc/stopnc: starts up (closes down)	/etc/stopnc: starts up (closes down) the file	error(2)
/etc/stopnc: starts up (closes down) the file	/etc/usam: initiate a UNIX server for a remote	error(3M)
/etc/usam: initiate a UNIX server for a remote	/etc/utab: neighbours known to the Newcastle	error(3M)
/etc/utab: neighbours known to the Newcastle	etext, edata: last locations in program.	error(3M)
etext, edata: last locations in program.	ethernet addresses.	error(3M)
ethernet addresses.	ethname: get name/ethernet-identification of	error(3M)
ethname: get name/ethernet-identification of	Euclidean distance function.	error(3M)
Euclidean distance function.	evaluate arguments as an expression.	error(3M)
evaluate arguments as an expression.	evaluation command.	error(3M)
evaluation command.	ex: text editor.	error(3M)
ex: text editor.	ex. vi:	error(3M)
ex. vi:	exor: run a program on another system.	error(3M)
exor: run a program on another system.	exec, execv, execl, execlp, execlvp: execute a file.	error(3M)
exec, execv, execl, execlp, execlvp: execute a file.	execl, execlp, execlvp: execute a file.	error(3M)
execl, execlp, execlvp: execute a file.	execute a file.	error(3M)
execute a file.	execute command.	error(3M)
execute command.	execute commands at a later time.	error(3M)
execute commands at a later time.	at(1)	error(3M)
at(1)	env: set environment for command	error(3M)
env: set environment for command	sleep: suspend	error(3M)
sleep: suspend	sleep: suspend	error(3M)
sleep: suspend	monitor: prepare	error(3M)
monitor: prepare	profil:	error(3M)
profil:	uux: unix to unix command	error(3M)
uux: unix to unix command	file. execl,	error(3M)
file. execl,	execl, execlp, execlvp: execute a	error(3M)
execl, execlp, execlvp: execute a	execlvp: execute a file.	error(3M)
execlvp: execute a file.	exercise link and unlink system calls.	error(3M)
exercise link and unlink system calls.	existing one.	error(3M)
existing one.	exit, _exit: terminate process.	error(3M)
exit, _exit: terminate process.	_exit: terminate process.	error(3M)
_exit: terminate process.	exp, dexp, cexp: Fortran exponential intrinsic	error(3M)
exp, dexp, cexp: Fortran exponential intrinsic	exp, log, log10, pow, sqrt: exponential, logarithm,	error(3M)
exp, log, log10, pow, sqrt: exponential, logarithm,	expand tabs to spaces, and vice versa.	error(3M)
expand tabs to spaces, and vice versa.	expand, unexpand: expand tabs to spaces, and vice	error(3M)
expand, unexpand: expand tabs to spaces, and vice	expire: remove outdated news articles.	error(3M)
expire: remove outdated news articles.	explain: print wordy sentences, interactive	error(3M)
explain: print wordy sentences, interactive	explicit Fortran type conversion. /idint, real,	error(3M)
explicit Fortran type conversion. /idint, real,	Exploring The Dungeons of Doom.	error(3M)
Exploring The Dungeons of Doom.	exponent.	error(3M)
exponent.	exponential intrinsic function.	error(3M)
exponential intrinsic function.	exponential, logarithm, power, square root	error(3M)
exponential, logarithm, power, square root	expr: evaluate arguments as an expression.	error(3M)
expr: evaluate arguments as an expression.	expression compile and match routines.	error(3M)
expression compile and match routines.	expression compile.	error(3M)
expression compile.	regexp: regular	error(3M)
regexp: regular	regcomp: regular	error(3M)
regcomp: regular		error(3M)

Permuted Index

regex, regcomp: regular	expression compile/execute.	regex(3X)
expr: evaluate arguments as an	expression.	expr(1)
efl:	Extended Fortran Language.	efl(1)
greek: graphics for the	extended TTY-37 type-box.	greek(7)
strings. xstr:	extract strings from C programs to implement shared	xstr(1)
	f77: Fortran 77 compiler.	f77(1)
functions. floor, ceil, fmod,	fabs: floor, ceiling, remainder, absolute value	floor(3M)
factor:	factor a number.	factor(1)
	factor: factor a number.	factor(1)
true,	false: provide truth values.	true(1)
checkall:	faster file system checking procedure.	checkall(8)
abort: generate an IOT	fault.	abort(3C)
	fclose, fflush: close or flush a stream.	fclose(3S)
	fcntl: file control.	fcntl(2)
	fcntl: file control options.	fcntl(7)
ecvt,	fcvt: output conversion.	ecvt(3C)
fopen, freopen,	fdopen: open a stream.	fopen(3S)
stctrl, stskip: special streamer	features, skip files.	stctrl(1)
col: filter reverse line	feeds.	col(1)
ferror,	feof, clearerr, fileno: stream status inquiries.	ferror(3S)
inquiries.	ferror, feof, clearerr, fileno: stream status	ferror(3S)
subroutines. dbminit,	fetch, store, delete, firstkey, nextkey: data base	dbm(3X)
system.	ff: list file names and statistics for a file	ff(8)
fclose,	fflush: close or flush a stream.	fclose(3S)
getc, getchar,	fgetc, getw: get character or word from stream.	getc(3S)
gets,	fgets: get a string from a stream.	gets(3S)
grep, egrep,	fgrep: search a file for a pattern.	grep(1)
umask: set	file-creation mode mask.	umask(1)
basename: strip	filename affixes.	basename(1)
ferror, feof, clearerr,	fileno: stream status inquiries.	ferror(3S)
acctcom: search and print process accounting	file(s).	acctcom(1)
compact, uncompact, ccat: compress and uncompress	files, and cat them.	compact(1)
backup.	filesave, tapesave: daily/weekly UNIX file system	filesave(8)
more, page: file perusal	filter for crt viewing.	more(1)
nl: line numbering	filter.	nl(1)
colort:	filter nroff output for CRT previewing.	colort(1)
col:	filter reverse line feeds.	col(1)
plot: graphics	filters.	plot(1G)
tplot: graphics	filters.	tplot(1G)
refer, lookbib:	find and insert literature references in documents.	refer(1)
find:	find files.	find(1)
	find: find files.	find(1)
hyphen:	find hyphenated words.	hyphen(1)
look:	find lines in a sorted list.	look(1)
ttyname, isatty:	find name of a terminal.	ttyname(3C)
lorder:	find ordering relation for an object library.	lorder(1)
spell, spellin, spellout:	find spelling errors.	spell(1)
binary, file. strings:	find the printable strings in a object, or other	strings(1)
ttyslot:	find the slot in the utmp file of the current user.	ttyslot(3C)
	finger: user information lookup program.	finger(1)
fold: fold long lines for	finite width output device.	fold(1)
dbminit, fetch, store, delete,	firstkey, nextkey: data base subroutines.	dbm(3X)
tee: pipe	fitting.	tee(1)
explicit Fortran type/ int, ifix, idint, real,	float, single, double, complex, decimal, ichar, char:	ftype(3f)
fp:	Floating Point on the CADMUS.	fp(3)
absolute value functions.	floor, ceil, fmod, fabs: floor, ceiling, remainder.	floor(3M)
functions. floor, ceil, fmod, fabs:	floor, ceiling, remainder, absolute value	floor(3M)
rxctrl:	sloppy disk manipulating program.	rxctrl(1)
ot, ox: TM 503/TM 603/TM 703 disk, TM 100-4	sloppy disk.	ot(4)
rx: RX01 or RX02	sloppy disk.	rx(4)
fclose, fflush: close or	flush a stream.	fclose(3S)
value functions. floor, ceil,	fmod, fabs: floor, ceiling, remainder, absolute	floor(3M)
	fmt: simple text formatter.	fmt(1)
device.	fold: fold long lines for finite width output	fold(1)
fold:	fold long lines for finite width output device.	fold(1)
vfont:	font formats for the Benson-Varian or Versatec.	vfont(5)
vfontinfo: inspect and print out information about	fonts.	vfontinfo(1)
	fopen, freopen, fdopen: open a stream.	fopen(3S)
	fork: create a new process.	fork(2)
dmesg: collect system diagnostic messages to	form error log.	dmesg(8)
indent: indent and	format a C program source.	indent(1)
acct: per-process accounting file	format.	acct(5)
ar: archive (library) file	format.	ar(5)
format: how to	format disks.	format(8)
dump, ddate: incremental dump	format.	dump(5)
	format: how to format disks.	format(8)
eqn, neqn, checkeq:	format mathematical text for nroff or troff.	eqn(1)
inode:	format of an inode.	inode(5)
core:	format of core image file.	core(5)

Permuted Index

cpio:	format of cpio archive.	cpio(5)
dir:	format of directories.	dir(5)
a.out:	format of programs and modules.	ahdr(5)
a.out:	format of programs and modules.	a.out(5)
scsfile:	format of SCCS file.	scsfile(5)
fs: file system	format of system volume.	fs(5)
fspec:	format specification in text files.	fspec(5)
tbl:	format tables for nroff or troff.	tbl(1)
troff, nroff: typeset or	format text.	troff(1)
vfont: font	formats for the Benson-Varian or Versatec.	vfont(5)
intro: introduction to file	formats.	intro(5)
utmp, wtmp: utmp and wtmp entry	formats.	utmp(5)
scanf, fscanf, sscanf:	formatted input conversion.	scanf(3S)
mm: print out documents	formatted with the MM macros.	mm(1)
fmt: simple text	formatting.	fmt(1)
printf, fprintf, sprintf: output	formatting documents.	printf(3S)
mptx: the macro package for	formatting a permuted index.	mptx(7)
check: disk checking and	formatting.	check(8)
mm: the MM macro package for	formatting documents.	mm(7)
mosd: the OSDD adapter macro package for	formatting documents.	mosd(7)
man: macros for	formatting entries in this manual.	man(7)
ms: macros for	formatting manuscripts.	ms(7)
me: macros for	formatting papers.	me(7)
f77:	Fortran 77 compiler.	f77(1)
abs, iabs, dabs, cabs, zabs:	Fortran absolute value.	abs(3f)
signal: specify	Fortran action on receipt of a system signal.	signal(3f)
acos, dacos:	Fortran arccosine intrinsic function.	acos(3f)
asin, dasin:	Fortran arcsine intrinsic function.	asin(3f)
atan2, datan2:	Fortran arctangent intrinsic function.	atan2(3f)
atan, datan:	Fortran arctangent intrinsic function.	atan(3f)
and, or, xor, not, lshift, rshift:	Fortran bitwise boolean functions.	bool(3f)
getarg: return	Fortran command-line argument.	getarg(3f)
log10, alog10, dlog10:	Fortran common logarithm intrinsic function.	log10(3f)
conjg, dconjg:	Fortran complex conjugate intrinsic function.	conjg(3f)
cos, dcos, ccos:	Fortran cosine intrinsic function.	cos(3f)
ratfor: rational	Fortran dialect.	ratfor(1)
getenv: return	Fortran environment variable.	getenv(3f)
exp, dexp, cexp:	Fortran exponential intrinsic function.	exp(3f)
cosh, dcosh:	Fortran hyperbolic cosine intrinsic function.	cosh(3f)
sinh, dsinh:	Fortran hyperbolic sine intrinsic function.	sinh(3f)
tanh, dtanh:	Fortran hyperbolic tangent intrinsic function.	tanh(3f)
aimag, dimag:	Fortran imaginary part of complex argument.	aimag(3f)
aint, dint:	Fortran integer part intrinsic function.	aint(3f)
efl: Extended	Fortran Language.	efl(1)
max, max0, amax0, max1, amax1, dmax1:	Fortran maximum-value functions.	max(3f)
min, min0, amin0, min1, amin1, dmin1:	Fortran minimum-value functions.	min(3f)
log, alog, dlog, clog:	Fortran natural logarithm intrinsic function.	log(3f)
anint, dnint, nint, idnint:	Fortran nearest integer functions.	round(3f)
abort: terminate	Fortran program.	abort(3f)
struct: structure	Fortran programs.	struct(1)
mod, amod, dmod:	Fortran remaindering intrinsic functions.	mod(3f)
sin, dsin, csin:	Fortran sine intrinsic function.	sin(3f)
sqrt, dsqrt, csqrt:	Fortran square root intrinsic function.	sqrt(3f)
len: return length of	Fortran string.	len(3f)
index: return location of	Fortran substring.	index(3f)
system: issue a shell command from	Fortran.	system(3f)
tan, dtan:	Fortran tangent intrinsic function.	tan(3f)
mclock: return	Fortran time accounting.	mclock(3f)
sign, isign, dsign:	Fortran transfer-of-sign intrinsic function.	sign(3f)
sngl, dble, cmplx, dcmplx, ichar, char: explicit	Fortran type conversion. /ifx, idint, real, float,	ftype(3f)
rand, srand:	Fortran uniform random-number generator.	rand(3f)
printf:	fp: Floating Point on the CADMUS.	fp(3f)
putc, putchar:	fprintf, sprintf: output formatters.	printf(3S)
puts:	fputc, putw: put character or word on a stream.	putc(3S)
C stack	fputs: put a string on a stream.	puts(3S)
	frame layout.	stack(5)
	fread, fwrite: buffered binary input/output.	fread(3S)
df: report number of	free disk blocks.	df(1)
df: report number of	free disk blocks.	df(8)
malloc:	free, realloc, calloc: main memory allocator.	malloc(3C)
fopen:	freopen, fdopen: open a stream.	fopen(3S)
hertz: get the line	frequency on the current machine.	hertz(2)
exponent.	frexp, ldexp, modf: split into mantissa and	frexp(3C)
colrm: remove columns	from a file.	colrm(1)
gets, fgets: get a string	from a stream.	gets(3S)
rmdbl: remove a delta	from an SCCS file.	rmdbl(1)
getopt: get option letter	from argv.	getopt(3C)
xstr: extract strings	from C programs to implement shared strings.	xstr(1)
read: read	from file.	read(2)

Permuted Index

system: issue a shell command	from Fortran.	system(3f)
from: who is my mail	from?.	from(1)
ncheck: generate names	from i-numbers.	ncheck(8)
nlist: get entries	from name list.	nlist(3C)
soelim: eliminate .so's	from nroff input.	soelim(1)
acctcms: command summary	from per-process accounting records.	acctcms(8)
getc, getchar, fgetc, getw: get character or word	from stream.	getc(3S)
getpw: get name	from UID.	getpw(3C)
	from: who is my mail from?.	from(1)
	fs: file system format of system volume.	fs(5)
	fsba: file system block analyzer.	fsba(8)
scanf, fscanf, sscanf: formatted input conversion.		scanf(3S)
checklist: list of file systems processed by interactive repair.	fsck.	checklist(5)
	fsck, fsck: file system consistency check and	fsck(8)
	fsdb: file system debugger.	fsdb(8)
	fseek, ftell, rewind: reposition a stream.	fseek(3S)
	fspec: format specification in text files.	fspec(5)
stat, fstat: get file status.		stat(2)
fseek, ftell, rewind: reposition a stream.		fseek(3S)
ftw: walk a file tree.		ftw(3C)
chfn: change full name of user.		chfn(1)
erf, erf: error function and complementary error function.		erf(3M)
structure. fuser: identify processes using a file or file		fuser(8)
fread, fwrite: buffered binary input/output.		fread(3S)
records. ftmp, wtmpfx: manipulate connect accounting		ftmp(8)
backgammon: the game.		backgammon(8)
startrek: THE game based on the t.v. series.		startrek(8)
moo: guessing game.		moo(8)
back: the game of backgammon.		back(8)
bj: the game of black jack.		bj(8)
craps: the game of craps.		craps(8)
reversi: reversi, a game of dramatic reversals.		reversi(8)
wump: the game of hunt-the-wumpus.		wump(8)
intro: introduction to games.		intro(8)
gamma: log gamma function.		gamma(3M)
	gamma: log gamma function.	gamma(3M)
send: gather files and submit RJE jobs.		send(1C)
abort: generate an IOT fault.		abort(3C)
newconf: generate configuration file and reconfigure MUNIX.		newconf(8)
makekey: generate encryption key.		makekey(8)
ctermid: generate file name for terminal.		ctermid(3S)
ncheck: generate names from i-numbers.		ncheck(8)
/mrand48, jrand48, srand48, seed48, lcong48: generate uniformly distributed pseudo-random/		drand48(3C)
lex: generator of lexical analysis programs.		lex(1)
rand, srand: random number generator.		rand(3C)
rand, srand: Fortran uniform random-number generator.		rand(3f)
gets, fgets: get a string from a stream.		gets(3S)
get: get a version of an SCCS file.		get(1)
ulimit: get and set user limits.		ulimit(2)
getc, getchar, fgetc, getw: get character or word from stream.		getc(3S)
nlist: get entries from name list.		nlist(3C)
umask: set and get file creation mask.		umask(2)
stat, fstat: get file status.		stat(2)
ustat: get file system statistics.		ustat(2)
get: get a version of an SCCS file.		get(1)
getgrent, getgrgid, getgrnam, setgrent, endgrent: get group file entry.		getgrent(3C)
getlogin: get login name.		getlogin(3C)
logname: get login name.		logname(1)
getpw: get name from UID.		getpw(3C)
system. uname, ethname: get name/ethernet-identification of current UNIX		uname(2)
unset: undo a previous get of an SCCS file.		unset(1)
getopt: get option letter from argv.		getopt(3C)
getpwent, getpwuid, getpwnam, setpwent, endpwent: get password file entry.		getpwent(3C)
getcwd: get path-name of current working directory.		getcwd(3C)
times: get process and child process times.		times(2)
getpid, getppid, getpgid: get process, process group, and parent process IDs.		getpid(2)
effective group/ getuid, geteuid, getgid, getegid: get real user, effective user, real group, and		getuid(2)
tty: get terminal name.		tty(1)
hertz: get the line frequency on the current machine.		hertz(2)
time: get time.		time(2)
getarg: return Fortran command-line argument.		getarg(3f)
getc, getchar, fgetc, getw: get character or word		getc(3S)
getchar, fgetc, getw: get character or word from		getc(3S)
getcwd: get path-name of current working directory.		getcwd(3C)
getegid: get real user, effective user, real group,		getuid(2)
getenv: return Fortran environment variable.		getenv(3f)
getenv: value for environment name.		getenv(3C)
geteuid, getgid, getegid: get real user, effective		getuid(2)
getgid, getegid: get real user, effective user,		getuid(2)
from stream. stream. getc,		
and effective group IDs. getuid, geteuid, getgid,		
user, real group, and effective group IDs. getuid,		
real group, and effective group/ getuid, geteuid,		

Permuted Index

get group file entry.	getgrnt, getgrgid, getgrnam, setgrnt, endgrnt:	getgrnt(3C)
file entry. getgrnt,	getgrgid, getgrnam, setgrnt, endgrnt: get group	getgrnt(3C)
getgrnt, getgrgid,	getgrnam, setgrnt, endgrnt: get group file entry.	getgrnt(3C)
	getlogin: get login name.	getlogin(3C)
	getopt: get option letter from argv.	getopt(3C)
	getopt: parse command options.	getopt(1)
	getpass: read a password.	getpass(3C)
parent process IDs. getpid,	getpgrp, getppid: get process, process group, and	getpid(2)
group, and parent process IDs.	getpid, getpgrp, getppid: get process, process	getpid(2)
process IDs. getpid, getpgrp,	getppid: get process, process group, and parent	getpid(2)
	getpw: get name from UID.	getpw(3C)
get password file entry.	getpwent, getpwuid, getpwnam, setpwent, endpwent:	getpwent(3C)
entry. getpwent, getpwuid,	getpwnam, setpwent, endpwent: get password file	getpwent(3C)
password file entry. getpwent,	getpwuid, getpwnam, setpwent, endpwent: get	getpwent(3C)
	gets, fgets: get a string from a stream.	gets(3S)
	getty: set terminal type, modes, speed, and line	getty(8)
	getty.	getty(5)
gettydefs: speed and terminal settings used by	getty.	gettydefs(5)
getty.	getuid, geteuid, getgid, getegid: get real user,	getuid(2)
effective user, real group, and effective group/	getutent, getutid, getutline, pututline, setutent,	getut(3C)
endutent, utmpname: access utmp file entry. getutent,	getutid, getutline, pututline, setutent, endutent,	getut(3C)
utmpname: access utmp file entry. getutent,	getutline, pututline, setutent, endutent, utmpname.	getut(3C)
access utmp file entry. getutent, getutid,	getw: get character or word from stream.	getc(3S)
getc, getchar, fgets,	gmtime, asctime, tzset: convert date and time to	ctime(3C)
ASCII, ctime, localtime,	goto.	setjmp(3C)
setjmp, longjmp: non-local	graph: draw a graph.	graph(1G)
	graph.	graph(1G)
graph: draw a	graph.	sag(1G)
sag: system activity	plot: graphics filters.	plot(1G)
plot:	plot: graphics filters.	tplot(1G)
tplot:	greek: graphics for the extended TTY-37 type-box.	greek(7)
greek:	plot: graphics interface.	plot(5)
plot:	plot: graphics interface subroutines.	plot(3X)
plot:	graphs, and slides.	mmt(1)
mmt, mvt: typeset documents, view	graphs.	mv(7)
mvt: a macro package for making view	greek: graphics for the extended TTY-37 type-box.	greek(7)
	grep, egrep, fgrep: search a file for a pattern.	grep(1)
getegid: get real user, effective user, real	group, and effective group IDs. /geteuid, getgid,	getuid(2)
getpid, getpgrp, getppid: get process, process	group, and parent process IDs.	getpid(2)
chown, chgrp: change owner or	group.	chown(1)
getgrgid, getgrnam, setgrnt, endgrnt: get	group file entry. getgrnt,	getgrnt(3C)
group:	group file.	group(5)
	group: group file.	group(5)
/etc/pwmap, /etc/groupmap: user and	group id mappings for the Newcastle Connection.	pwmap(5N)
setpgrp: set process	group ID.	setpgrp(2)
id: print user and	group IDs and names.	id(1)
user, effective user, real group, and effective	group IDs. /geteuid, getgid, getegid: get real	getuid(2)
setuid, setgid: set user and	group IDs.	setuid(2)
newgrp: log in to a new	group.	newgrp(1)
chown: change owner and	group of a file.	chown(2)
kill: send a signal to a process or a	group of processes.	kill(2)
make: maintain, update, and regenerate	groups of programs.	make(1)
	grpck: password/group file checkers.	pwck(8)
pwck,	signal: software signals.	signal(3C)
signal,	guess the word.	hangman(8)
hangman:	guessing game.	moo(8)
moo:	handle special functions of HP 2640 and 2621-series	hp(1)
terminals. hp:	hangman: guess the word.	hangman(8)
	hash search tables.	hsearch(3C)
hsearch, hcreate, hdestroy: manage	hsearch, hcreate, hdestroy: manage hash search tables.	hsearch(3C)
hsearch,	hdestroy: manage hash search tables.	hsearch(3C)
hsearch, hcreate,	help: ask for help.	help(1)
	help.	help(1)
help: ask for	hertz: get the line frequency on the current	hertz(2)
machine.	xd: hexadecimal dump.	xd(1)
xd:	hier: file system hierarchy.	hier(7)
hier: file system	hierarchy.	hier(7)
	hk: RK611/RK06, RK07 moving-head disk.	hk(4)
rl:	hl: RL01/RL02 moving-head disk.	rl(4)
hp: handle special functions of	HP 2640 and 2621-series terminals.	hp(1)
2621-series terminals.	hp: handle special functions of HP 2640 and	hp(1)
	hp: RP04/05/08, RM02/03 moving-head disk.	hp(4)
tables.	hsearch, hcreate, hdestroy: manage hash search	hsearch(3C)
wump: the game of	hunt-the-wumpus.	wump(8)
cosh, dcosh: Fortran	hyperbolic cosine intrinsic function.	cosh(3f)
sinh, cosh, tanh:	hyperbolic functions.	sinh(3M)
sinh, dsinh: Fortran	hyperbolic sine intrinsic function.	sinh(3f)
tanh, dtanh: Fortran	hyperbolic tangent intrinsic function.	tanh(3f)
	hyphen: find hyphenated words.	hyphen(1)

Permuted Index

hyphen: find	hyphenated words.	hyphen(1)
abs,	hypot: Euclidean distance function.	hypot(3M)
rje: RJE (Remote Job Entry) to	iabs, dabs, cabs, zabs: Fortran absolute value.	abs(3f)
/idint, real, float, singl, dble, cmplx, dcmplx,	iargc: Number of command-line arguments.	iargc(3f)
/etc/pwmap, /etc/groupmap: user and group	IBM.	rje(8)
/etc/setugi: alter user	ichar, char: explicit Fortran type conversion.	ftype(3f)
setpgrp: set process group	id mappings for the Newcastle Connection.	pwmap(5N)
issue: issue	id of a UNIX server.	setugi(8N)
fuser:	id: print user and group IDs and names.	id(1)
what:	ID.	setpgrp(2)
ichar, char: explicit Fortran type/ int, ifix,	identification file.	issue(5)
anint, dnint, nint,	identify processes using a file or file structure.	fuser(8)
id: print user and group	identify SCCS files.	what(1)
get process, process group, and parent process	idint, real, float, singl, dble, cmplx, dcmplx,	ftype(3f)
effective user, real group, and effective group	idnint: Fortran nearest integer functions.	round(3f)
setuid, setgid: set user and group	IDs and names.	id(1)
dcmplx, ichar, char: explicit Fortran type/ int,	IDs, getpid, getpgrp, getppid:	getpid(2)
core: format of core	IDs, /getuid, getgid, getegid: get real user,	getuid(2)
aimag, dimag: Fortran	IDs.	setuid(2)
xstr: extract strings from C programs to	ifix, idint, real, float, singl, dble, cmplx,	ftype(3f)
which: locate a program file	ifprolog: The prolog interpreter system.	ifprolog(1)
dump, ddate:	image file.	core(5)
dump:	imaginary part of complex argument.	aimag(3f)
restor:	implement shared strings.	xstr(1)
indent:	including aliases and paths (csh only).	which(1)
tgetnum, tgetflag, tgetstr, tgoto, tputs: terminal	incremental dump format.	dump(5)
mptx: the macro package for formatting a permuted	incremental file system dump.	dump(8)
ptx: permuted	incremental file system restore.	restor(8)
inittab: script for the	indent and format a C program source.	indent(1)
process. /etc/NCsetup:	indent: indent and format a C program source.	indent(1)
init, telinit: process control	independent operation routines. tgetent,	termcap(3)
brc, bcheckrc, rc, powerfail: system	index.	mptx(7)
/etc/usam:	index.	ptx(1)
popen, pclose:	index: return location of Fortran substring.	index(3f)
clri: clear	inews: submit news articles.	inews(1)
inode: format of an	init process.	inittab(5)
scanf, fscanf, sscanf: formatted	init, telinit: process control initialization.	init(8)
soelim: eliminate .so's from nroff	initialise the Newcastle Connection tables in a	NCsetup(8N)
ungetc: push character back into	initialization.	init(8)
fread, fwrite: buffered binary	initialization shell scripts.	brc(8)
stdio: standard buffered	initiate a UNIX server for a remote client.	usam(8N)
fferror, feof, clearerr, fileno: stream status	initiate I/O to/from a process.	popen(3S)
uustat: uucp status	inittab: script for the init process.	inittab(5)
refer, lookbib: find and	i-node.	clri(8)
vfontinfo:	inode: format of an inode.	inode(5)
install:	inode.	inode(5)
uucp: uucp	input conversion.	scanf(3S)
mail.nc: mail(1) using the Newcastle Connection	input.	soelim(1)
learn: computer aided	input stream.	ungetc(3S)
dcmplx, ichar, char: explicit Fortran type/	input/output.	fread(3S)
abs:	input/output package.	stdio(3S)
anint, dnint, nint, idnint: Fortran nearest	inquiries.	fferror(3S)
aint, dint: Fortran	inquiry and job control.	uustat(1C)
l3tol, ltol3: convert between 3-byte	insert literature references in documents.	refer(1)
ltol3: convert between 3-byte integers and long	inspect and print out information about fonts.	vfontinfo(1)
bcopy:	install commands.	install(8)
fsck, dfsck: file system consistency check and	install: install commands.	install(8)
rjstat: RJE status report and	installation made easy.	uucp(8)
diction, explain: print wordy sentences,	instead of uucp.	mail(1N)
bbp: Basic Block Port	instruction about UNIX.	learn(1)
kl: KL-11 or DL-11 asynchronous	int, ifix, idint, real, float, singl, dble, cmplx,	ftype(3f)
lbp: LBP-10 Laser Beam Printer	integer absolute value.	abs(3C)
plot: graphics	integer functions.	round(3f)
sbp: Simplified Basic Block Port	integer part intrinsic function.	aint(3f)
st: SCT11 Streamer	integers and long integers.	l3tol(3C)
plot: graphics	integers. l3tol,	l3tol(3C)
termio: general terminal	interactive block copy.	bcopy(8)
tm,ts: TM-11/TU-10 magtape interface, TS-11 magtape	interactive repair.	fsck(8)
	interactive status console.	rjstat(1C)
	interactive thesaurus for diction.	diction(1)
	Interface.	bbp(4)
	interface.	kl(4)
	Interface.	lbp(4)
	interface.	plot(5)
	Interface.	sbp(4)
	interface.	st(4)
	interface subroutines.	plot(3X)
	interface.	termio(4)
	interface.	tm(4)

Permuted Index

nfcomment: a user	interface to the notesfile system.	nfcomment(3)
tm,ts: TM-11/TU-10 magtape	interface, TS-11 magtape interface.	tm(4)
tty: controlling terminal	interface.	tty(4)
spline:	interpolate smooth curve.	spline(1G)
basic: Basic	Interpreter.	basic(1)
pti: phototypesetter	interpreter.	pti(1)
sno: SNOBOL	interpreter.	sno(1)
ifprolog: The prolog	interpreter system.	ifprolog(1)
csh: a shell (command	interpreter) with C-like syntax.	csh(1)
pipe: create an	interprocess channel.	pipe(2)
configuration information: table of	interrupt vector and device addresses.	confinfo(4)
sleep: suspend execution for an	interval.	sleep(1)
sleep: suspend execution for	interval.	sleep(3C)
	intro: introduction to commands.	intro(1)
	intro: introduction to file formats.	intro(5)
	intro: introduction to games.	intro(8)
	intro: introduction to miscellany.	intro(7)
	intro: introduction to special files.	intro(4)
	intro: introduction to subroutines and libraries.	intro(3)
	intro: introduction to system calls and error	intro(2)
	intro: introduction to system maintenance	intro(8)
numbers.	i-numbers.	ncheck(8)
procedures.	I/O to/from a process.	popen(3S)
ncheck: generate names from	ioctl: control device.	ioctl(2)
popen, pclose: initiate	IOT fault.	abort(3C)
	isalnum, isspace, ispunct, isprint, isgraph, /	ctype(3C)
abort: generate an	isalpha, isupper, islower, isdigit, isxdigit,	ctype(3C)
isalnum, isspace, ispunct, isprint, isgraph, /	isascii: classify characters. /isxdigit, isalnum,	ctype(3C)
isspace, ispunct, isprint, isgraph, isctrl,	isatty: find name of a terminal.	ttyname(3C)
ttyname,	isctrl, isascii: classify characters. /isxdigit,	ctype(3C)
isalnum, isspace, ispunct, isprint, isgraph,	isdigit, isxdigit, isalnum, isspace, ispunct,	ctype(3C)
isprint, isgraph, / isalpha, isupper, islower,	isgraph, isctrl, isascii: classify characters.	ctype(3C)
/isxdigit, isalnum, isspace, ispunct, isprint,	isgn, dsign: Fortran transfer-of-sign intrinsic	sign(3f)
function. sign,	islower, isdigit, isxdigit, isalnum, isspace,	ctype(3C)
ispunct, isprint, isgraph, / isalpha, isupper,	isprint, isgraph, isctrl, isascii: classify /	ctype(3C)
/isdigit, isxdigit, isalnum, isspace, ispunct,	ispunct, isprint, isgraph, isctrl, isascii: /	ctype(3C)
/islower, isdigit, isxdigit, isalnum, isspace,	isspace, ispunct, isprint, isgraph, isctrl, /	ctype(3C)
/isupper, islower, isdigit, isxdigit, isalnum,	issue a shell command.	system(3S)
system:	issue a shell command from Fortran.	system(3f)
system:	issue identification file.	issue(5)
issue:	issue: issue identification file.	issue(5)
isspace, ispunct, isprint, isgraph, / isalpha,	isupper, islower, isdigit, isxdigit, isalnum,	ctype(3C)
isgraph, / isalpha, isupper, islower, isdigit,	isxdigit, isalnum, isspace, ispunct, isprint,	ctype(3C)
news: print news	items.	news(1)
	j0, j1, jn, y0, y1, yn: Bessel functions.	bessel(3M)
j0.	j1, jn, y0, y1, yn: Bessel functions.	bessel(3M)
bj: the game of black	jack.	bj(6)
j0, j1.	jn, y0, y1, yn: Bessel functions.	bessel(3M)
	join: relational database operator.	join(1)
drand48, erand48, lrand48, nrand48, mrand48,	rand48, srand48, seed48, lcong48: generate /	drand48(3C)
msgs: system messages and	junk mail program.	msgs(1)
whatconf: what device drivers are in an unix	kernel.	whatconf(8)
makekey: generate encryption	key.	makekey(8)
keycap:	keyboard capability data base.	keycap(5)
	keycap: keyboard capability data base.	keycap(5)
apropos: locate commands by	keyword lookup.	apropos(1)
killall:	kill all active processes.	killall(8)
processes.	kill: send a signal to a process or a group of	kill(2)
	kill: terminate a process.	kill(1)
	killall: kill all active processes.	killall(8)
	kl: KL-11 or DL-11 asynchronous interface.	kl(4)
kl:	KL-11 or DL-11 asynchronous interface.	kl(4)
mem.	kmem: core memory.	mem(4)
quiz: test your	knowledge.	quiz(8)
ls, ll,	l, lr, lf, lx: list contents of directory.	ls(1)
long integers.	l3tol, ltol3: convert between 3-byte integers and	l3tol(3C)
a64l,	l64a: convert between long and base-64 ASCII.	a64l(3C)
volcopy, labelit: copy file systems with	label checking.	volcopy(8)
volcopy,	labelit: copy file systems with label checking.	volcopy(8)
awk: pattern scanning and processing	language.	awk(1)
bc: arbitrary-precision arithmetic	language.	bc(1)
efl: Extended Fortran	language.	efl(1)
cpp: the C	language preprocessor.	cpp(1)
shell, the standard/restricted command programming	language. sh, rsh:	sh(1)
lbp: LBP-10	Laser Beam Printer Interface.	lbp(4)
prtacct, runacct, / chargefee, ckpacct, dodisk,	lastlogin, monacct, nulladm, prttmp, prdaily,	acctsh(8)
C stack frame	layout.	stack(5)
	lbp: LBP-10 Laser Beam Printer Interface.	lbp(4)
ltroff, vtroff: troff to the CANON	LBP or the Versatec V80.	ltroff(1)

Permuted Index

lbp:	LBP-10 Laser Beam Printer Interface.	lbp(4)
cc:	C compiler.	cc(1)
lrand48, mrand48, jrand48, srand48, seed48,	lcong48: generate uniformly distributed / /rand48,	drand48(3C)
ld:	loader.	ld(1)
ldexp:	ldexp, modf: split into mantissa and exponent.	ldexp(3C)
learn:	computer aided instruction about UNIX.	learn(1)
leave:	leave.	leave(1)
leave:	remind you when you have to leave.	leave(1)
len:	return length of Fortran string.	len(3f)
length of Fortran string.		len(3f)
letter from argv.		getopt(3C)
lex:	generator of lexical analysis programs.	lex(1)
ls, ll, l, lr,	lexical analysis programs.	lex(1)
lf, lx:	list contents of directory.	ls(1)
libraries.		intro(3)
libraries.		ranlib(1)
ranlib:	convert archives to random (library) file format.	ar(5)
ar:	archive	order(1)
lorder:	find ordering relation for an object	ar(1)
ar:	archive and	ulimit(2)
ulimit:	get and set user	getty(8)
getty:	set terminal type, modes, speed, and	col(1)
col:	filter reverse	hertz(2)
hertz:	get the	line(1)
line:	read one	nl(1)
nl:	line numbering filter.	cut(1)
cut:	cut out selected fields of each	lpd(8)
lpd:	line printer daemon.	lp(4)
lp:	parallel	lpctrl(1)
lpctrl:	set options on the parallel	lpr(1)
lpr:	line printer spooler.	line(1)
line:	read one line.	lsearch(3C)
lsearch:	linear search and update.	head(1)
head:	give first few	link(8)
link, unlink:	exercise	link(2)
link:	link to a file.	cp(1)
cp, ln, mv:	copy, link or move files.	link(2)
link:	link to a file.	link(8)
calls:	link, unlink: exercise link and unlink system	lint(1)
lint:	a C program verifier.	ls(1)
ls, ll, l, lr, lf, lx:	list contents of directory.	ff(8)
ff:	list file names and statistics for a file system.	look(1)
look:	find lines in a sorted	nlist(3C)
nlist:	get entries from name	nm(1)
nm:	print name	checklist(5)
checklist:	list of file systems processed by fsck.	users(1)
users:	compact	users(1)
users:	compact	users(1)
cref:	make cross-reference	ceref(1)
xref:	cross-reference	xref(1)
xargs:	construct argument	xargs(1)
refer, lookbib:	find and insert	refer(1)
ls,	ll, l, lr, lf, lx: list contents of directory.	ls(1)
cp,	ln, mv: copy, link or move files.	cp(1)
ld:	loader.	ld(1)
time to ASCII, ctime,	localtime, gmtime, asctime, tzset: convert date and	ctime(3C)
(csh only), which:	locate a program file including aliases and paths	which(1)
apropos:	locate commands by keyword lookup.	apropos(1)
whereis:	locate source, binary, and or manual for program.	whereis(1)
index:	return	index(3f)
end, etext, edata:	last	end(3C)
plock:	lock process, text, or data in memory.	plock(2)
lock:	reserve a terminal.	lock(1)
log, alog, dlog, clog:	Fortran natural logarithm	log(3f)
log, dmesg:		dmesg(8)
gamma:	log gamma function.	gamma(3M)
newgrp:	log in to a new group.	newgrp(1)
power, square root functions, exp,	log, log10, pow, sqrt: exponential, logarithm,	exp(3M)
intrinsic function.	log10, alog10, dlog10: Fortran common logarithm	log10(3f)
square root functions, exp, log,	log10, pow, sqrt: exponential, logarithm, power,	exp(3M)
log10, alog10, dlog10:	Fortran common	log10(3f)
log, alog, dlog, clog:	Fortran natural	log(3f)
exp, log, log10, pow, sqrt:	exponential,	exp(3M)
getlogin:	get	getlogin(3C)
logname:	get	logname(1)
cuserid:	character	cuserid(3S)
logname:	login name of user.	logname(3X)
passwd:	change	passwd(1)
login:	sign on.	login(1)
profile:	setting up an environment at	profile(5)
login:	time.	login(1)
logname:	get login name.	logname(1)

Permuted Index

setjmp,	logfile: login name of user.	logfile(3X)
documents. refer,	longjmp: non-local goto.	setjmp(3C)
apropos: locate commands by keyword	lookbib: find and insert literature references in	refer(1)
finger: user information	lookup.	apropos(1)
library.	lookup program.	finger(1)
nice, nohup: run a command at	lorder: find ordering relation for an object.	lorder(1)
	low priority.	nice(1)
	lp: parallel line printer.	lp(4)
	lpctrl: set options on the parallel line printer.	lpctrl(1)
	lpd: line printer daemon.	lpd(8)
	lpr: line printer spooler.	lpr(1)
ls, ll, l,	lr, lf, lx: list contents of directory.	ls(1)
seed48, lcong48: generate/ drand48, erand48,	lrand48, nrand48, mrand48, jrand48, srand48,	drand48(3C)
	ls, ll, l, lr, lf, lx: list contents of directory.	ls(1)
	lsearch: linear search and update.	lsearch(3C)
	lseek: move read/write file pointer.	lseek(2)
and, or, xor, not,	lshift, rshift: Fortran bitwise boolean functions.	bool(3f)
integers. l3tol,	lto13: convert between 3-byte integers and long	l3tol(3C)
Versatec V80.	ltroff, vtroff: troff to the CANON LBP or the	ltroff(1)
ls, ll, l, lr, lf,	lx: list contents of directory.	ls(1)
hertz: get the line frequency on the current	m4: macro processor.	m4(1)
mptx: the	machine.	hertz(2)
mm: the MM	macro package for formatting a permuted index.	mptx(7)
mosd: the OSDD adapter	macro package for formatting documents.	mm(7)
mv: a	macro package for formatting documents.	mosd(7)
m4:	macro package for making view graphs.	mv(7)
mm: print out documents formatted with the MM	macro processor.	m4(1)
mt:	macros.	mm(1)
tm,ts: TM-11/TU-10 magtape interface, TS-11	magnetic tape manipulating program.	mt(1)
tm,ts: TM-11/TU-10	magtape interface.	tm(4)
from: who is my	magtape interface, TS-11 magtape interface.	tm(4)
prmail: print out	mail from?.	from(1)
mail, rmail: send mail to users or read	mail in the post office.	prmail(1)
msgs: system messages and junk	mail.	mail(1)
recnews: receive unprocessed articles via	mail program.	msgs(1)
recnews: receive unprocessed articles via	mail.	recnews(1)
	mail.	recnews(8)
	mail, rmail: send mail to users or read mail.	mail(1)
sendnews: send news articles via	mail.	sendnews(8)
mail, rmail: send	mail to users or read mail.	mail(1)
uurec: receive processed news articles via	mail.	uurec(8)
uucp.. mail.nc:	mail(1) using the Newcastle Connection instead of	mail(1N)
instead of uucp..	mail.nc: mail(1) using the Newcastle Connection	mail(1N)
malloc, free, realloc, calloc:	main memory allocator.	malloc(3C)
programs. make:	maintain, update, and regenerate groups of	make(1)
ar: archive and library	maintainer.	ar(1)
intro: introduction to system	maintenance procedures.	intro(8)
delta:	make a delta (change) to an SCCS file.	delta(1) -
mkdir:	make a directory.	mkdir(1)
mknod:	make a directory, or a special or ordinary file.	mknod(2)
/etc/mksys:	make a remote system node.	mksys(8N)
mktemp:	make a unique file name.	mktemp(3C)
cref:	make cross-reference listing.	cref(1)
banner:	make long posters.	banner(8)
programs. make:	make: maintain, update, and regenerate groups of	make(1)
script:	make typescript of terminal session.	script(1)
allocator.	makekey: generate encryption key.	makekey(8)
	malloc, free, realloc, calloc: main memory	malloc(3C)
	man: macros for formatting entries in this manual.	man(7)
	man, manprog: print entries in this manual.	man(1)
	man: print sections of this manual.	man(1b)
tsearch, tdelete, twalk:	manage binary search trees.	tsearch(3C)
hsearch, hcreate, hdestroy:	manage hash search tables.	hsearch(3C)
ftwtmp, wtmpfix:	manipulate connect accounting records.	ftwtmp(8)
mt: magnetic tape	manipulating program.	mt(1)
rxctrl: floppy disk	manipulating program.	rxctrl(1)
man,	manprog: print entries in this manual.	man(1)
frexp, ldexp, modf: split into	mantissa and exponent.	frexp(3C)
catman: create the cat files for the	manual.	catman(8)
whereis: locate source, binary, and or	manual for program.	whereis(1)
man, manprog: print entries in this	manual.	man(1)
man: print sections of this	manual.	man(1b)
man: macros for formatting entries in this	manuscripts.	man(7)
ms: macros for formatting	map of ASCII character set.	ms(7)
ascii:	mappings for the Newcastle Connection.	ascii(7)
/etc/pwmap, /etc/groupmap: user and group id	mask.	pwmap(5N)
umask: set file-creation mode	mask.	umask(1)
umask: set and get file creation	mask.	umask(2)
mkstr: create an error message file by	massaging C source.	mkstr(1)

Permuted Index

regexp: regular expression compile and
 eqn, neqn, checkeq: format
 maximum-value functions.
 maximum-value functions. max,
 functions. max, max0, amax0,
 max, max0, amax0, max1, amax1, dmax1: Fortran
 bed, ppt: convert to antique
 malloc, free, realloc, calloc: main
 mem, kmem: core
 plock: lock process, text, or data in
 sort: sort or
 acctmerg:
 lines of one file. paste:
 mkstr: create an error
 minimum-value functions.
 minimum-value functions. min,
 functions. min, min0, amin0,
 min, min0, amin0, min1, amin1, dmin1: Fortran
 a68:
 newsoutput: notesfile utility programs.
 file.
 source.
 mm: the
 mmchek: check usage of
 mm: print out documents formatted with the
 macros.
 delimiters.
 slides.
 functions.
 chmod: change
 umask: set file-creation
 chmod: change
 getty: set terminal type.
 bs: a compiler/interpreter for
 frexp, ldexp,
 utime: set file access and
 long: system calls
 long: standard procedures
 touch: update date last
 a.out: format of programs and
 a.out: format of programs and
 runacct, chargefee, ckpacct, dodisk, lastlogin,
 uusub:
 documents.
 curses: screen functions with "optimal" cursor
 mount:
 mount, umount:
 setmnt: establish
 mnttab:
 mvdrr:
 cp, ln, mv: copy, link or
 lseek:
 hk: RK611/RK06, RK07
 hp: RP04/05/06, RM02/03
 rl, hl: RL01/RL02
 rm: RM02/03/05
 index.
 generate/ drand48, erand48, lrand48, nrand48,
 dz, dh: DZ-11, DH-11 asynchronous
 generate configuration file and reconfigure
 match routines.
 mathematical text for nroff or troff.
 matherr: error-handling function.
 max, max0, amax0, max1, amax1, dmax1: Fortran
 max0, amax0, max1, amax1, dmax1: Fortran
 max1, amax1, dmax1: Fortran maximum-value
 maximum-value functions.
 mclock: return Fortran time accounting.
 med: screen editor.
 media.
 mem, kmem: core memory.
 memory allocator.
 memory.
 memory.
 merge files.
 merge or add total accounting files.
 merge same lines of several files or subsequent
 msg: permit or deny messages.
 message file by messaging C source.
 min, min0, amin0, min1, amin1, dmin1: Fortran
 min0, amin0, min1, amin1, dmin1: Fortran
 min1, amin1, dmin1: Fortran minimum-value
 minimum-value functions.
 MIT assembler.
 mkdir: make a directory.
 mkfs: construct a file system.
 mkfnf, rmfnf, nfxmit, nfcv, nfarchive, newsinput,
 mknod: build special file.
 mknod: make a directory, or a special or ordinary
 mkstr: create an error message file by messaging C
 mktemp: make a unique file name.
 MM macro package for formatting documents.
 mm macros and eqn delimiters.
 MM macros.
 mm: print out documents formatted with the MM
 mm: the MM macro package for formatting documents.
 mmchek: check usage of mm macros and eqn
 mnt, mvt: typeset documents, view graphs, and
 mnttab: mounted file system table.
 mod, amod, dmod: Fortran remaindering intrinsic
 mode.
 mode mask.
 mode of file.
 modes, speed, and line discipline.
 modest-sized programs.
 modf: split into mantissa and exponent.
 modification times.
 modified for long arguments.
 modified for long arguments.
 modified of a file.
 modules.
 modules.
 monacct, nulladm, pretmp, prdaily, prtacct,
 monitor: prepare execution profile.
 monitor uucp network.
 moo: guessing game.
 more, page: file perusal filter for crt viewing.
 mosd: the OSDD adapter macro package for formatting
 motion.
 mount a file system.
 mount and dismount file system.
 mount: mount a file system.
 mount table.
 mount, umount: mount and dismount file system.
 mounted file system table.
 move a directory.
 move files.
 move read/write file pointer.
 moving-head disk.
 moving-head disk.
 moving-head disk.
 moving-head disk.
 mptx: the macro package for formatting a permuted
 mrnd48, jrnd48, srnd48, seed48, lcong48:
 ms: macros for formatting manuscripts.
 msgs: system messages and junk mail program.
 mt: magnetic tape manipulating program.
 multiplexers.
 MUNIX. newconf:
 regexp(7)
 eqn(1)
 matherr(3M)
 max(3f)
 max(3f)
 max(3f)
 max(3f)
 mclock(3f)
 med(1)
 bcd(8)
 mem(4)
 malloc(3C)
 mem(4)
 plock(2)
 sort(1)
 acctmerg(8)
 paste(1)
 msg(1)
 mkstr(1)
 min(3f)
 min(3f)
 min(3f)
 min(3f)
 a68(1)
 mkdrr(1)
 mkfs(8)
 notes(8)
 mknod(8)
 mknod(2)
 mkstr(1)
 mktemp(3C)
 mm(7)
 mmchek(1)
 mm(1)
 mm(1)
 mm(7)
 mmchek(1)
 mnt(1)
 mnttab(5)
 mod(3f)
 chmod(1)
 umask(1)
 chmod(2)
 getty(8)
 bs(1)
 frexp(3C)
 utime(2)
 long(2)
 long(3C)
 touch(1)
 ahdr(5)
 a.out(5)
 acctsh(8)
 monitor(3C)
 uusub(8)
 moo(8)
 more(1)
 mosd(7)
 curses(3)
 mount(2)
 mount(8)
 mount(2)
 setmnt(8)
 mount(8)
 mnttab(5)
 mvdrr(8)
 cp(1)
 lseek(2)
 hk(4)
 hp(4)
 rl(4)
 rm(4)
 mptx(7)
 drand48(3C)
 ms(7)
 msgs(1)
 mt(1)
 dz(4)
 newconf(8)

Permuted Index

cp, ln,	mv: a macro package for making view graphs.	mv(7)
	mv: copy, link or move files.	cp(1)
	mvdir: move a directory.	mvdir(8)
mmt,	mvt: typeset documents, view graphs, and slides.	mmt(1)
pipe: Pipes and	named pipes.	pipe(4)
system. uname, ethname: get	name/ethernet-identification of current UNDX	uname(2)
log, alog, dlog, clog: Fortran	natural logarithm intrinsic function.	log(3f)
	ncheck: generate names from i-numbers.	ncheck(8)
anint, dnint, nint, idnint: Fortran	nearest integer functions.	round(3f)
/etc/utab:	neighbours known to the Newcastle Connection.	utab(5N)
or troff. eqn,	neqn, checkeq: format mathematical text for nroff	eqn(1)
eqnchar: special character definitions for eqn and	neqn.	eqnchar(7)
news: USENET	network news article, utility files.	news(5)
uusub: monitor uuap	network.	uusub(8)
mail.nc: mail(1) using the	Newcastle Connection instead of uuap.	mail(1N)
/etc/groupmap: user and group id mappings for the	Newcastle Connection. /etc/pwmap.	pwmap(5N)
/etc/NCsetup: initialise the	Newcastle Connection tables in a process.	NCsetup(8N)
/etc/utab: neighbours known to the	Newcastle Connection.	utab(5N)
reconfigure MUNDX.	newconf: generate configuration file and	newconf(8)
	newgrp: log in to a new group.	newgrp(1)
news: USENET network	news article, utility files.	news(5)
expire: remove outdated	news articles.	expire(8)
inews: submit	news articles.	inews(1)
postnews: submit	news articles.	postnews(1)
readnews: read	news articles via mail.	readnews(1)
sendnews: send	news articles via mail.	sendnews(8)
uurec: receive processed	news.	uurec(8)
checknews: check to see if user has	news items.	checknews(1)
news: print	news: print news items.	news(1)
notes, autoseq, nfpip, nfprint, nfstats: a	news system.	notes(1)
	news: USENET network news article, utility files.	news(5)
mknf, rmnf, nfzmit, nfrev, nfarchive,	newsinput, newsoutput: notesfile utility programs.	notes(8)
mknf, rmnf, nfzmit, nfrev, nfarchive, newsinput,	newsoutput: notesfile utility programs.	notes(8)
checknews(1).	newsr: information file for readnews(1) and	newsr(5)
dbminit, fetch, store, delete, firstkey,	nextkey: data base subroutines.	dbm(3X)
programs. mknf, rmnf, nfzmit, nfrev,	nfarchive, newsinput, newsoutput: notesfile utility	notes(8)
system.	nfcomment: a user interface to the notesfile	nfcomment(3)
notes, autoseq,	nfpip, nfprint, nfstats: a news system.	notes(1)
notes, autoseq, nfpip,	nfprint, nfstats: a news system.	notes(1)
utility programs. mknf, rmnf, nfzmit,	nfrev, nfarchive, newsinput, newsoutput: notesfile	notes(8)
notes, autoseq, nfpip, nfprint,	nfstats: a news system.	notes(1)
notesfile utility programs. mknf, rmnf,	nfzmit, nfrev, nfarchive, newsinput, newsoutput:	notes(8)
	nice: change priority of a process.	nice(2)
	nice, nohup: run a command at low priority.	nice(1)
alter priority of running process by changing	nice. renice:	renice(8)
anint, dnint,	nint, idnint: Fortran nearest integer functions.	round(3f)
	nl: line numbering filter.	nl(1)
	nlist: get entries from name list.	nlist(3C)
/etc/mksys: make a remote system	nm: print name list.	nm(1)
nice,	node.	mksys(8N)
setjmp, longjmp:	nohup: run a command at low priority.	nice(1)
functions. and, or, xor,	non-local goto.	setjmp(3C)
system.	not, lshift, rshift: Fortran bitwise boolean	bool(3f)
nfcomment: a user interface to the	notes, autoseq, nfpip, nfprint, nfstats: a news	notes(1)
nfzmit, nfrev, nfarchive, newsinput, newsoutput:	notesfile system.	nfcomment(3)
lcong48: generate/ drand48, erand48, lrand48,	notesfile utility programs. mknf, rmnf,	notes(8)
soelim: eliminate .so's from	nrand48, mrand48, jrand48, srand48, seed48,	drand48(3C)
eqn, neqn, checkeq: format mathematical text for	nroff input.	soelim(1)
tbl: format tables for	nroff or troff.	eqn(1)
colert: filter	nroff or troff.	tbl(1)
troff,	nroff output for CRT previewing.	colert(1)
checknr: check	nroff: typeset or format text.	troff(1)
deroff: remove	nroff/troff files.	checknr(1)
null: the	nroff/troff, tbl, and eqn constructs.	deroff(1)
chargefee, ckpacct, dodisk, lastlogin, monacct,	null file.	null(4)
nl: line	null: the null file.	null(4)
size: size of an	nulladm, prttmp, prdaily, prtacct, runacct,/	acctsh(8)
lorder: find ordering relation for an	numbering filter.	nl(1)
strings: find the printable strings in a	object file.	size(1)
od :	object library.	lorder(1)
	object, or other binary, file.	strings(1)
pack : packs or unpacks many files <-->	octal dump.	od(1)
a program file including aliases and paths (csh	od : octal dump.	od(1)
fopen, freopen, fdopen:	one..	pack(1)
dup: duplicate an	only), which: locate	which(1)
open:	open a stream.	fopen(3S)
	open file descriptor.	dup(2)
	open for reading or writing.	open(2)

Permuted Index

[illegible]

Permuted Index

root functions. exp, log, log10,	postnews: submit news articles.	postnews(1)
exp, log, log10, pow, sqrt: exponential, logarithm,	pow, sqrt: exponential, logarithm, power, square	exp(3M)
brc, bcheckrc, rc,	power, square root functions.	exp(3M)
bcd,	powerfail: system initialization shell scripts.	brc(8)
/ckpacct, dodisk, lastlogin, monacct, nulladm,	ppt: convert to antique media.	bcd(6)
/dodisk, lastlogin, monacct, nulladm, pretmp,	pr: print files.	pr(1)
monitor:	pretmp, prdaily, prtacct, runacct, shutacct, /	acctsh(8)
prep:	prdaily, prtacct, runacct, shutacct, startup, /	acctsh(8)
cpp: the C language	prep: prepare text for statistical processing.	prep(1)
colcrt: filter troff output for CRT	prepare execution profile.	monitor(3C)
unget: undo a	prepare text for statistical processing.	prep(1)
types:	preprocessor.	cpp(1)
lpd: line	previewing.	colcrt(1)
lbp: LBP-10 Laser Beam	previous get of an SCCS file.	unget(1)
lp: parallel line	primitive system data types.	types(7)
lpctrl: set options on the parallel line	printer daemon.	lpd(8)
lpr: line	Printer Interface.	lbp(4)
vp: Versatec	printer.	lp(4)
nice, nohup: run a command at low	printer.	lpctrl(1)
nice: change	printer spooler.	lpr(1)
renice: alter	printer-plotter.	vp(4)
exit, _exit: terminate	printf, sprintf, sprintf: output formatters.	printf(3S)
fork: create a new	priority.	nice(1)
inittab: script for the init	priority of a process.	nice(2)
kill: terminate a	priority of running process by changing nice.	renice(8)
initialise the Newcastle Connection tables in a	prmail: print out mail in the post office.	prmail(1)
nice: change priority of a	process.	exit(2)
popen, pelose: initiate I/O to/from a	process.	fork(2)
getpid, getpgid, getppid: get	process.	inittab(5)
plock: lock	process.	kill(1)
wait: await completion of	process. /etc/NCsetup:	NCsetup(8N)
checklist: list of file systems	process.	nice(2)
uurec: receive	process.	popen(3S)
kill: send a signal to a process or a group of	process, process group, and parent process IDs.	getpid(2)
killall: kill all active	process, text, or data in memory.	plock(2)
fuser: identify	process.	wait(1)
prep: prepare text for statistical	processed by fack.	checklist(5)
shutdown: terminate all	processed news articles via mail.	uurec(8)
m4: macro	processes.	kill(2)
alarm: set a	processes.	killall(8)
prof: display	processes using a file or file structure.	fuser(8)
monitor: prepare execution	processing.	prep(1)
profil: execution time	processing.	shutdown(8)
ah, rsh: shell, the standard/restricted command	processor.	m4(1)
ifprolog: The	process's alarm clock.	alarm(2)
arithmetic:	prof: display profile data.	prof(1)
true, false:	profil: execution time profile.	profil(2)
/lastlogin, monacct, nulladm, pretmp, prdaily,	profile data.	prof(1)
seed48, lcong48: generate uniformly distributed	profile.	monitor(3C)
ungetc:	profile.	profil(2)
on a stream.	profile: setting up an environment at login time.	profile(5)
stream. putc,	programming language.	sh(1)
utmp file entry. getutent, getutid, getutline,	prolog interpreter system.	ifprolog(1)
putc, putchar, fputc,	provide drill in number facts.	arithmetic(6)
qsart:	provide truth values.	true(1)
generator.	prs: print an SCCS file.	prs(1)
ranlib: convert archives to	prtacct, runacct, shutacct, startup, turnacct: /	acctsh(8)
	ps: process status.	ps(1)
	pseudo-random numbers. /mrand48, jrand48, srand48,	drand48(3C)
	pstat: print system facts.	pstat(8)
	pti: phototypesetter interpreter.	pti(1)
	ptrace: process trace.	ptrace(2)
	ptx: permuted index.	ptx(1)
	push character back into input stream.	ungetc(3S)
	putc, putchar, fputc, putw: put character or word	putc(3S)
	putchar, fputc, putw: put character or word on a	putc(3S)
	putpwent: write password file entry.	putpwent(3C)
	puts, fputs: put a string on a stream.	puts(3S)
	pututline, setutent, endutent, utmpname: access	getut(3C)
	putw: put character or word on a stream.	putc(3S)
	pwck, grpck: password/group file checkers.	pwck(8)
	pwd: working directory name.	pwd(1)
	qsort: quicker sort.	qsort(3C)
	quicker sort.	qsort(3C)
	quiz: test your knowledge.	quiz(8)
	quot: summarize file system ownership.	quot(8)
	rand, srand: Fortran uniform random-number	rand(3f)
	rand, srand: random number generator.	rand(3C)
	random libraries.	ranlib(1)

Permuted Index

rand, srand:	random number generator.	rand(3C)
rand, srand: Fortran uniform	random-number generator.	rand(3f)
	ranlib: convert archives to random libraries.	ranlib(1)
	ratfor: rational Fortran dialect.	ratfor(1)
	ratfor: rational Fortran dialect.	ratfor(1)
brc, bcheckrc,	rc, powerfail: system initialization shell scripts.	brc(8)
getpass:	read a password.	getpass(3C)
read:	read from file.	read(2)
mail, rmail: send mail to users or	read mail.	mail(1)
readnews:	read news articles.	readnews(1)
line:	read one line.	line(1)
	read: read from file.	read(2)
open: open for	reading or writing.	open(2)
	readnews: read news articles.	readnews(1)
newsrsc: information file for	readnews(1) and checknews(1).	newsrsc(5)
lseek: move	read/write file pointer.	lseek(2)
char: explicit Fortran type/	real, float, singl, dble, cmplx, dcmplx, ichar,	fctype(3f)
int, iflx, idint,	realloc, calloc: main memory allocator.	malloc(3C)
malloc, free,	receipt of a signal.	signal(2)
signal: specify what to do upon	receipt of a system signal.	signal(3f)
signal: specify Fortran action on	receive processed news articles via mail.	uurec(8)
uurec:	receive unprocessed articles via mail.	recnews(1)
recnews:	receive unprocessed articles via mail.	recnews(8)
recnews:	recnews: receive unprocessed articles via mail.	recnews(1)
	recnews: receive unprocessed articles via mail.	recnews(8)
	reconfigure MUNIX.	newconf(8)
newconf: generate configuration file and	records. acctcms:	acctcms(8)
command summary from per-process accounting	records.	fwtmp(8)
fwtmp, wtmpfix: manipulate connect accounting	ed:	ed(1)
	references in documents.	refer(1)
refer, lookbib: find and insert literature	refer, lookbib: find and insert literature	refer(1)
	references in documents.	refer(1)
	reform: reformat text file.	reform(1)
reform:	reformat text file.	reform(1)
	regcomp: regular expression compile.	regcomp(1)
regex,	regcomp: regular expression compile/execute.	regex(3X)
make: maintain, update, and	regenerate groups of programs.	make(1)
	regex, regcomp: regular expression compile/execute.	regex(3X)
routines.	regexp: regular expression compile and match	regexp(7)
regexp:	regular expression compile and match routines.	regexp(7)
regcomp:	regular expression compile.	regcomp(1)
regex, regcomp:	regular expression compile/execute.	regex(3X)
comm: select or	reject lines common to two sorted files.	comm(1)
lorder: find ordering	relation for an object library.	lorder(1)
unify	relational data base system.	unify(1)
join:	relational database operator.	join(1)
strip: remove symbols and	relocation bits.	strip(1)
floor, ceil, fmod, fabs: floor, ceiling,	remainder, absolute value functions.	floor(3M)
mod, amod, dmod: Fortran	remaindering intrinsic functions.	mod(3f)
leave:	remind you when you have to leave.	leave(1)
calendar:	reminder service.	calendar(1)
/etc/usam: initiate a UNIX server for a	remote client.	usam(8N)
usrv: UNIX server for a	remote client.	usrv(8N)
/etc/mkalias: create an alias to a	remote file.	mkalias(8N)
rje: RJE	(Remote Job Entry) to IBM.	rje(8)
/etc/rmsys: remove a	remote system name.	rmsys(8N)
/etc/mksys: make a	remote system node.	mksys(8N)
unite: enable a	remote user to access the local system.	unite(8N)
rmddel:	remove a delta from an SCCS file.	rmddel(1)
/etc/rmsys:	remove a remote system name.	rmsys(8N)
colrm:	remove columns from a file.	colrm(1)
unlink:	remove directory entry.	unlink(2)
rm, rmdir:	remove files or directories.	rm(1)
deroff:	remove nroff/troff, tbl, and eqn constructs.	deroff(1)
expire:	remove outdated news articles.	expire(8)
strip:	remove symbols and relocation bits.	strip(1)
changing nice.	renice: alter priority of running process by	renice(8)
file system consistency check and interactive	repair. fsck, dfsck:	fsck(8)
uniq: report	repeated lines in a file.	uniq(1)
yes: be	repetitively affirmative.	yes(1)
sar: system activity	reporter.	sar(1)
fseek, ftell, rewind:	reposition a stream.	fseek(3S)
lock:	reserve a terminal.	lock(1)
restor: incremental file system	restor: incremental file system restore.	restor(8)
getarg:	return Fortran command-line argument.	restor(8)
getenv:	return Fortran environment variable.	getarg(3f)
mclock:	return Fortran time accounting.	getenv(3f)
len:	return length of Fortran string.	mclock(3f)
index:	return location of Fortran substring.	len(3f)
		index(3f)

Permuted Index

stat: data	returned by stat system call.	stat(7)
reversi: reversi, a game of dramatic	rev: reverse lines of a file.	rev(1)
col: filter	reversals.	reversi(8)
rev:	reverse line feeds.	col(1)
reversi:	reverse lines of a file.	rev(1)
fseek, ftell,	reversi, a game of dramatic reversals.	reversi(8)
creat: create a new file or	reversi: reversi, a game of dramatic reversals.	reversi(8)
send: gather files and submit	rewind: reposition a stream.	fseek(3S)
rje:	rewrite an existing one.	creat(2)
rje:	RJE jobs.	send(1C)
rje:	RJE (Remote Job Entry) to IBM.	rje(8)
rje:	RJE (Remote Job Entry) to IBM.	rje(8)
rjstat:	RJE status report and interactive status console.	rjstat(1C)
console:	rjstat: RJE status report and interactive status	rjstat(1C)
rk: RK-11/RK03 or	rk: RK-11/RK03 or RK05 disk.	rk(4)
hk: RK811/RK08,	RK05 disk.	rk(4)
rk:	RK07 moving-head disk.	hk(4)
hk:	RK-11/RK03 or RK05 disk.	rk(4)
rl, hl:	RK811/RK08, RK07 moving-head disk.	hk(4)
hp: RP04/05/08,	rl, hl: RL01/RL02 moving-head disk.	rl(4)
rm:	RL01/RL02 moving-head disk.	rl(4)
mail:	rm: RM02/03/05 moving-head disk.	rm(4)
rm:	rm, rmdir: remove files or directories.	rm(1)
newsoutput: notesfile utility programs. mknf,	RM02/03 moving-head disk.	hp(4)
chroot: change	RM02/03/05 moving-head disk.	rm(4)
chroot: change	rmail: send mail to users or read mail.	mail(1)
pow, sqrt: exponential, logarithm, power, square	rmdel: remove a delta from an SCCS file.	rmdel(1)
sqrt, dsqrt, csqrt: Fortran square	rmdir: remove files or directories.	rm(1)
regex: regular expression compile and match	rmnf, nfmf, nfmf, nfmf, nfmf, nfmf, nfmf,	notes(8)
rgoto, tputs: terminal independent operation	rogue: Exploring The Dungeons of Doom.	rogue(8)
hp:	root directory.	chroot(2)
programming language. sh,	root directory for a command.	chroot(8)
and, or, xor, not, lshift,	root functions. exp, log, log10,	exp(3M)
nice, nohup:	root intrinsic function.	sqrt(3f)
exec:	root functions.	regexp(7)
runacct:	root functions. tgetent, tgetnum, tgetflag, tgetstr,	termcap(3)
/monacct, nulladm, prttmp, prdaily, prtacct,	RP04/05/08, RM02/03 moving-head disk.	hp(4)
renice: alter priority of	rsh: shell, the standard/restricted command	sh(1)
rx:	rshift: Fortran bitwise boolean functions.	bool(3f)
rx: RX01 or	run: run a command at low priority.	nice(1)
rx: RX01 or	run: run a program on another system.	exec(1N)
rx: RX01 or	runacct: run daily accounting.	runacct(8)
rx: RX01 or	runacct: run daily accounting.	runacct(8)
rx: RX01 or	runacct, shutacct, startup, turnacct: shell/	acctsh(8)
rx: RX01 or	running process by changing nice.	renice(8)
rx: RX01 or	rx: RX01 or RX02 floppy disk.	rx(4)
rx: RX01 or	rx: RX01 or RX02 floppy disk.	rx(4)
rx: RX01 or	rx: RX02 floppy disk.	rx(4)
rx: RX01 or	rxctrl: floppy disk manipulating program.	rxctrl(1)
rx: RX01 or	sa, accton: system accounting.	sa(8)
rx: RX01 or	sa1, sa2, sadc: system activity report package.	sar(8)
rx: RX01 or	sa2, sadc: system activity report package.	sar(8)
rx: RX01 or	sact: print current SCCS file editing activity.	sact(1)
rx: RX01 or	sadc: system activity report package.	sar(8)
rx: RX01 or	sag: system activity graph.	sag(1G)
rx: RX01 or	sar: system activity reporter.	sar(1)
rx: RX01 or	sbp: Simplified Basic Block Port Interface.	sbp(4)
rx: RX01 or	sbrk: change data segment space allocation.	brk(2)
rx: RX01 or	scanf, fscanf, sscanf: formatted input conversion.	scanf(3S)
rx: RX01 or	scanner.	bfs(1)
rx: RX01 or	scanning and processing language.	awk(1)
rx: RX01 or	SCCS delta.	cdc(1)
rx: RX01 or	SCCS deltas.	comb(1)
rx: RX01 or	SCCS file.	delta(1)
rx: RX01 or	SCCS file editing activity.	sact(1)
rx: RX01 or	SCCS file.	get(1)
rx: RX01 or	SCCS file.	prs(1)
rx: RX01 or	SCCS file.	rmdel(1)
rx: RX01 or	SCCS file.	scsdiff(1)
rx: RX01 or	SCCS file.	scsfile(5)
rx: RX01 or	SCCS file.	unget(1)
rx: RX01 or	SCCS file.	val(1)
rx: RX01 or	SCCS files.	admin(1)
rx: RX01 or	SCCS files.	what(1)
rx: RX01 or	scsdiff: compare two versions of an SCCS file.	scsdiff(1)
rx: RX01 or	scsfile: format of SCCS file.	scsfile(5)
rx: RX01 or	screen.	clear(1)
rx: RX01 or	screen editor.	emacs(1)
rx: RX01 or	screen editor.	med(1)

Permuted Index

curses:	screen functions with "optimal" cursor motion.	curses(3)
ex. vi:	screen oriented (visual) display editor based on	vi(1)
inittab:	script for the init process.	inittab(5)
	script: make typescript of terminal session.	script(1)
rc, powerfail: system initialization shell	scripts. brk, bcheckrc.	brk(8)
st:	SCT11 Streamer interface.	st(4)
	sdiff: side-by-side difference program.	sdiff(1)
grep, egrep, fgrep:	search a file for a pattern.	grep(1)
acctcom:	search and print process accounting file(s).	acctcom(1)
lsearch: linear	search and update.	lsearch(3C)
bsearch: binary	search.	bsearch(3C)
hsearch, hcreate, hdestroy: manage hash	search tables.	hsearch(3C)
tsearch, tdelete, twalk: manage binary	search trees.	tsearch(3C)
man: print	sections of this manual.	man(1b)
	sed: stream editor.	sed(1)
/rand48, nrand48, mrand48, jrand48, srand48,	seed48, lcong48: generate uniformly distributed/	drand48(3C)
brk, sbrk: change data	segment space allocation.	brk(2)
comm:	select or reject lines common to two sorted files.	comm(1)
cut: cut out	selected fields of each line of a file.	cut(1)
kill:	send a signal to a process or a group of processes.	kill(2)
	send: gather files and submit RJE jobs.	send(1C)
mail, rmail:	send mail to users or read mail.	mail(1)
sendnews:	send news articles via mail.	sendnews(8)
	sendnews: send news articles via mail.	sendnews(8)
diction, explain: print wordy	sentences, interactive thesaurus for diction.	diction(1)
startrek: THE game based on the t.v.	series.	startrek(8)
/etc/usam: initiate a UNIX	server for a remote client.	usam(8N)
usrv: UNIX	server for a remote client.	usrv(8N)
/etc/setugi: alter user id of a UNIX	server.	setugi(8N)
/etc/stopnc: starts up (closes down) the file	server spawner. /etc/startnc,	startnc(8N)
script: make typescript of terminal	session.	script(1)
	setbuf: assign buffering to a stream.	setbuf(3S)
setuid,	setgid: set user and group IDs.	setuid(2)
getgrent, getgrgid, getgrnam,	setgrent, endgrent: get group file entry.	getgrent(3C)
	setjmp, longjmp: non-local goto.	setjmp(3C)
crypt,	setkey, encrypt: DES encryption.	crypt(3C)
	setmnt: establish mount table.	setmnt(8)
getpwent, getpwuid, getpwnam;	setpgrp: set process group ID.	setpgrp(2)
profile:	setpwent, endpwent: get password file entry.	getpwent(3C)
gettydefs: speed and terminal	setting up an environment at login time.	profile(5)
	settings used by getty.	gettydefs(5)
entry. getutent, getutid, getutline, pututline,	setuid, setgid: set user and group IDs.	setuid(2)
programming language.	setutent, endutent, utmpname: access utmp file	getut(3C)
xstr: extract strings from C programs to implement	sh, rsh: shell, the standard/restricted command	sh(1)
system: issue a	shared strings.	xstr(1)
system: issue a	shell command.	system(3S)
csd: a	shell command from Fortran.	system(3f)
prtacct, runacct, shutacct, startup, turnacct:	shell (command interpreter) with C-like syntax.	csh(1)
brk, bcheckrc, rc, powerfail: system initialization	shell procedures for accounting. /prtmp, prdaily,	acctsh(8)
language. sh, rsh:	shell scripts.	brk(8)
/nulladm, prtmp, prdaily, prtacct, runacct,	shell, the standard/restricted command programming	sh(1)
	shutacct, startup, turnacct: shell procedures for/	acctsh(8)
	shutdown: terminate all processing.	shutdown(8)
sdiff:	side-by-side difference program.	sdiff(1)
intrinsic function.	sign, isign, dsign: Fortran transfer-of-sign	sign(3f)
login:	sign on.	login(1)
pause: suspend process until	signal.	pause(2)
signal: specify what to do upon receipt of a	signal.	signal(2)
specify Fortran action on receipt of a system	signal. signal:	signal(3f)
signal.	signal: specify Fortran action on receipt of a	signal(3f)
signal.	signal: specify what to do upon receipt of a	signal(2)
kill: send a	signal to a process or a group of processes.	kill(2)
ssignal, gsignal: software	signals.	ssignal(3C)
fmt:	simple text formatter.	fmt(1)
sbp:	Simplified Basic Block Port Interface.	sbp(4)
tc: phototypesetter	simulator.	tc(1)
trigonometric functions.	sin, cos, tan, asin, acos, atan, atan2:	trig(3M)
	sin, dsin, csin: Fortran sine intrinsic function.	sin(3f)
sin, dsin, csin: Fortran	sine intrinsic function.	sin(3f)
sinh, dsinh: Fortran hyperbolic	sine intrinsic function.	sinh(3f)
	sinh, cosh, tanh: hyperbolic functions.	sinh(3M)
function.	sinh, dsinh: Fortran hyperbolic sine intrinsic	sinh(3f)
size:	size of an object file.	size(1)
	size: size of an object file.	size(1)
stctrl, stskip: special streamer features,	skip files.	stctrl(1)
	sleep: suspend execution for an interval.	sleep(1)
mmt, mvt: typeset documents, view graphs, and	sleep: suspend execution for interval.	sleep(3C)
ttyslot: find the	slides.	mmt(1)
	slot in the utmp file of the current user.	ttyslot(3C)

Permuted Index

spline: interpolate
 Fortran type/ int, ifix, idint, real, float,
 sno:
 SNOBOL interpreter.
 ssignal, gsignal:
 sort:
 qsort: quicker
 tsort: topological
 comm: select or reject lines common to two
 look: find lines in a
 soelim: eliminate
 whereis: locate
 indent: indent and format a C program
 mkstr: create an error message file by massaging C
 brk, sbrk: change data segment
 expand, unexpand: expand tabs to
 starts up (closes down) the file server
 fspec: format
 getty: set terminal type, modes,
 gettydefs:
 spell,
 spell, spellin, spellout: find
 spell, spellin,
 split:
 csplit: context
 frexp, ldexp, modf:
 uuclean: uucp
 lpr: line printer
 printf, fprintf,
 function.
 functions: exp, log, log10, pow,
 log10, pow, sqrt: exponential, logarithm, power,
 sqrt, dsqrt, csqrt: Fortran
 rand,
 rand,
 /rand48, lrand48, nrand48, mrand48, jrand48,
 scanf, fscanf,
 C
 stksiz: set
 boot:
 stdio:
 long:
 sh, rsh: shell, the
 /etc/startnc, /etc/stopnc:
 /pretmp, prdaily, prtacct, runacct, shutacct,
 stat: data returned by
 prep: prepare text for
 ft: list file names and
 ustat: get file system
 rjstat: RJE status report and interactive
 ferror, feof, clearerr, fileno: stream
 uustat: uucp
 ps: process
 rjstat: RJE
 stat, fstat: get file
 files.
 wait: wait for child process to
 subroutines. dbm, fetch,
 strlen, strchr, strrchr, strpbrk, strspn, strcspn,
 /strncat, strcmp, strncmp, strcpy, strncpy, strlen,
 strrchr, strpbrk, strspn, / strcat, strncat,
 strspn, strcspn, / strcat, strncat, strcmp, strncmp,
 strncpy, strlen, strchr, strrchr, strpbrk, strspn,
 sed:
 fclose, fflush: close or flush a
 smooth curve.
 singl, dble, cmplx, dcmplx, ichar, char: explicit
 sno: SNOBOL interpreter.
 SNOBOL interpreter.
 soelim: eliminate .so's from nroff input.
 software signals.
 sort or merge files.
 sort.
 sort: sort or merge files.
 sort.
 sorted files.
 sorted list.
 .so's from nroff input.
 source, binary, and or manual for program.
 source.
 source.
 space allocation.
 spaces, and vice versa.
 spawner. /etc/startnc, /etc/stopnc:
 specification in text files.
 speed, and line discipline.
 speed and terminal settings used by getty.
 spell, spellin, spellout: find spelling errors.
 spellin, spellout: find spelling errors.
 spelling errors.
 spellout: find spelling errors.
 spline: interpolate smooth curve.
 split a file into pieces.
 split.
 split into mantissa and exponent.
 split: split a file into pieces.
 spool directory clean-up.
 spooler.
 sprintf: output formatters.
 sqrt, dsqrt, csqrt: Fortran square root intrinsic
 sqrt: exponential, logarithm, power, square root
 square root functions. exp, log,
 square root intrinsic function.
 srand: Fortran uniform random-number generator.
 srand: random number generator.
 srand48, seed48, lcong48: generate uniformly/
 sscanf: formatted input conversion.
 ssignal, gsignal: software signals.
 st: SCT11 Streamer interface.
 stack frame layout.
 stacksize.
 standalone startup program.
 standard buffered input/output package.
 standard procedures modified for long arguments.
 standard/restricted command programming language.
 startrek: THE game based on the t.v. series.
 starts up (closes down) the file server spawner.
 startup program.
 startup, turnacct: shell procedures for accounting.
 stat: data returned by stat system call.
 stat, fstat: get file status.
 stat system call.
 statistical processing.
 statistics for a file system.
 statistics.
 status console.
 status inquiries.
 status inquiry and job control.
 status.
 status report and interactive status console.
 status.
 stctrl, stskip: special streamer features, skip
 stdio: standard buffered input/output package.
 stime: set time.
 stksiz: set stacksize.
 stop or terminate.
 store, delete, firstkey, nextkey: data base
 strcat, strncat, strcmp, strncmp, strcpy, strncpy,
 strchr, strrchr, strpbrk, strspn, strcspn, strtok:
 strcmp, strncmp, strcpy, strncpy, strlen, strchr,
 strcpy, strncpy, strlen, strchr, strrchr, strpbrk,
 strcspn, strtok: string operations. /strcpy,
 stream editor.
 stream.
 spline(1G)
 ftype(3f)
 sno(1)
 sno(1)
 soelim(1)
 ssignal(3C)
 sort(1)
 qsort(3C)
 sort(1)
 tsort(1)
 comm(1)
 look(1)
 soelim(1)
 whereis(1)
 indent(1)
 mkstr(1)
 brk(2)
 expand(1)
 startnc(8N)
 fspec(5)
 getty(8)
 gettydefs(5)
 spell(1)
 spell(1)
 spell(1)
 spline(1G)
 split(1)
 csplit(1)
 frexp(3C)
 split(1)
 uuclean(8)
 lpr(1)
 printf(3S)
 sqrt(3f)
 exp(3M)
 exp(3M)
 sqrt(3f)
 rand(3f)
 rand(3C)
 drand48(3C)
 scanf(3S)
 ssignal(3C)
 st(4)
 stack(5)
 stksiz(1)
 boot(8)
 stdio(3S)
 long(3C)
 sh(1)
 startrek(8)
 startnc(8N)
 boot(8)
 acctsh(8)
 stat(7)
 stat(2)
 stat(7)
 prep(1)
 ft(8)
 ustat(2)
 rjstat(1C)
 ferror(3S)
 uustat(1C)
 ps(1)
 rjstat(1C)
 stat(2)
 stctrl(1)
 stdio(3S)
 stime(2)
 stksiz(1)
 wait(2)
 dbm(3X)
 string(3C)
 string(3C)
 string(3C)
 string(3C)
 string(3C)
 sed(1)
 fclose(3S)

Permuted Index

fopen, freopen, fdopen: open a	stream.	fopen(3S)
fseek, ftell, rewind: reposition a	stream.	fseek(3S)
getchar, fgetc, getw: get character or word from	stream. getc,	getc(3S)
gets, fgets: get a string from a	stream.	gets(3S)
putchar, fputc, putw: put character or word on a	stream. fputc,	putc(3S)
puts, fputs: put a string on a	stream.	puts(3S)
setbuf: assign buffering to a	stream.	setbuf(3S)
ferror, feof, clearerr, fileno:	stream status inquiries.	ferror(3S)
ungetc: push character back into input	stream.	ungetc(3S)
stcctl, stskip: special	streamer features, skip files.	stcctl(1)
st: SCT11	Streamer interface.	st(4)
gets, fgets: get a	string from a stream.	gets(3S)
len: return length of Fortran	string.	len(3f)
puts, fputs: put a	string on a stream.	puts(3S)
strchr, strchr, strpbrk, strspn, strtok:	string operations. /strcpy, strncpy, strlen,	string(3C)
other binary, file.	strings: find the printable strings in a object, or	strings(1)
strings. xstr: extract	strings from C programs to implement shared	xstr(1)
strings: find the printable	strings in a object, or other binary, file.	strings(1)
extract strings from C programs to implement shared	strings. xstr:	xstr(1)
basename:	strip filename affixes.	basename(1)
streat, strncat, stremp, strncmp, strcpy, strncpy,	strip: remove symbols and relocation bits.	strip(1)
strchr, strchr, strpbrk, strspn, strscn, / strcat,	strlen, strchr, strchr, strpbrk, strspn, strscn, /	string(3C)
strpbrk, strspn, strscn, / strcat, strncat, stremp,	strncat, stremp, strncmp, strcpy, strncpy, strlen,	string(3C)
strscn, / strcat, strncat, stremp, strncmp, strcpy,	strncmp, strcpy, strncpy, strlen, strchr, strchr,	string(3C)
strncmp, strcpy, strncpy, strlen, strchr, strchr,	strcpy, strlen, strchr, strchr, strpbrk, strspn,	string(3C)
/stremp, strncmp, strcpy, strncpy, strlen, strchr,	strpbrk, strspn, strscn, strtok: string / /stremp,	string(3C)
/strcpy, strncpy, strlen, strchr, strchr, strpbrk,	strchr, strpbrk, strspn, strscn, strtok: string /	string(3C)
strlen, strchr, strchr, strpbrk, strspn, strscn,	strspn, strscn, strtok: string operations.	string(3C)
struct:	strtok: string operations. /strcpy, strncpy,	string(3C)
fuser: identify processes using a file or file	struct: structure Fortran programs.	struct(1)
document.	structure.	struct(1)
inews:	stty: set the options for a terminal.	fuser(8)
postnews:	style : analyze surface characteristics of a	stty(1)
send: gather files and	su: become super-user or another user.	style(1)
intro: introduction to	submit news articles.	su(1)
fetch, store, delete, firstkey, nextkey: data base	submit RJE jobs.	inews(1)
plot: graphics interface	subroutines and libraries.	postnews(1)
paste: merge same lines of several files or	subroutines. dbmunit,	send(1C)
index: return location of Fortran	subroutines.	intro(3)
sum:	subsequent lines of one file.	dbm(3X)
du:	substring.	plot(3X)
quot:	sum and count blocks in a file.	paste(1)
acctcms: command	sum: sum and count blocks in a file.	index(3f)
sync: update the	summarize disk usage.	sum(1)
sync: update	super-block.	sum(1)
su: become	super-block.	du(1)
style: analyze	super-user or another user.	quot(8)
sleep:	summary from per-process accounting records.	acctcms(8)
sleep:	super block.	sync(8)
pause:	super-block.	sync(2)
swab:	super-user or another user.	su(1)
strip: remove	surface characteristics of a document.	style(1)
csn: a shell (command interpreter) with C-like	suspend execution for an interval.	sleep(1)
messages. perror, perror, sys_errlist,	suspend execution for interval.	sleep(3C)
mnttab: mounted file system	suspend process until signal.	pause(2)
/etc/map_port_eadr:	swab: swap bytes.	swab(3C)
configuration information:	swab: swap bytes.	swab(3C)
setmnt: establish mount	symbols and relocation bits.	strip(1)
hsearch, hcreate, hdestroy: manage hash search	sync: update super-block.	sync(2)
tbl: format	sync: update the super block.	sync(8)
/etc/NCsetup: initialise the Newcastle Connection	syntax.	csn(1)
tabs: set	sys_errlist, sys_nerr, errno: system error	perror(3C)
expand, unexpand: expand	sys_nerr, errno: system error messages.	perror(3C)
ctags: create a	table.	mnttab(5)
functions. sin, cos,	table of ethernet addresses.	ethmap(5N)
tan, dtan: Fortran	table of interrupt vector and device addresses.	confinfo(4)
tanh, dtanh: Fortran hyperbolic	table.	setmnt(8)
	tables.	hsearch(3C)
	tables for nroff or troff.	tbl(1)
	tables in a process.	NCsetup(8N)
	tabs on a terminal.	tabs(1)
	tabs: set tabs on a terminal.	tabs(1)
	tabs to spaces, and vice versa.	expand(1)
	tags file.	ctags(1)
	tail: deliver the last part of a file.	tail(1)
	tan, asin, acos, atan, atan2: trigonometric	trig(3M)
	tan, dtan: Fortran tangent intrinsic function.	tan(3f)
	tangent intrinsic function.	tan(3f)
	tangent intrinsic function.	tanh(3f)

Permuted Index

function.	tanh, dtanh: Fortran hyperbolic tangent intrinsic	tanh(3f)
sinh, cosh.	tanh: hyperbolic functions.	sinh(3M)
tar:	tape archiver.	tar(1)
dumpdir: print the names of files on a dump	tape.	dumpdir(8)
mt: magnetic	tape manipulating program.	mt(1)
filesave,	tapesave: daily/weekly UNIX file system backup.	filesave(8)
	tar: tape archiver.	tar(1)
deroff: remove nroff/troff,	tbl, and eqn constructs.	deroff(1)
	tbl: format tables for nroff or troff.	tbl(1)
	tc: phototypesetter simulator.	tc(1)
tsearch,	tdelete, twalk: manage binary search trees.	tsearch(3C)
	tee: pipe fitting.	tee(1)
tk: paginator for the	Tektronix 4014.	tk(1)
init,	telinit: process control initialization.	init(8)
tmpfile: create a	temporary file.	tmpfile(3S)
tmpnam: create a name for a	temporary file.	tmpnam(3S)
	term: conventional names.	term(7)
termcap:	termcap: terminal capability data base.	termcap(5)
ctermid: generate file name for	terminal capability data base.	termcap(5)
tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs:	terminal.	ctermid(3S)
termio: general	terminal independent operation routines.	termcap(3)
tty: controlling	terminal interface.	termio(4)
lock: reserve a	terminal interface.	tty(4)
tty: get	terminal.	lock(1)
clear: clear	terminal name.	tty(1)
script: make typescript of	terminal screen.	clear(1)
gettydefs: speed and	terminal session.	script(1)
stty: set the options for a	terminal settings used by getty.	gettydefs(5)
tabs: set tabs on a	terminal.	stty(1)
ttyname, isatty: find name of a	terminal.	tabs(1)
getty: set	terminal.	ttyname(3C)
handle special functions of HP 2640 and 2621-series	terminal type, modes, speed, and line discipline.	getty(8)
kill:	terminals. hp:	hp(1)
shutdown:	terminate a process.	kill(1)
abort:	terminate all processing.	shutdown(8)
exit, _exit:	terminate Fortran program.	abort(3f)
wait: wait for child process to stop or	terminate process.	exit(2)
	terminate.	wait(2)
quiz:	termio: general terminal interface.	termio(4)
d:	test: condition evaluation command.	test(1)
d:	test your knowledge.	quiz(8)
ed, red:	text database functions.	d(1)
ex:	text database functions.	d(3)
reform: reformat	text editor.	ed(1)
fspec: format specification in	text editor.	ex(1)
eqn, neqn, checkeq: format mathematical	text file.	reform(1)
prep: prepare	text files.	fspec(5)
fnt: simple	text for nroff or troff.	eqn(1)
plock: lock process.	text for statistical processing.	prep(1)
troff, nroff: typeset or format	text formatter.	fnt(1)
terminal independent operation routines. tgetent, tgetnum,	text, or data in memory.	plock(2)
independent operation routines. tgetent,	text.	troff(1)
independent operation routines. tgetent,	tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs:	termcap(3)
operation routines. tgetent, tgetnum, tgetflag,	tgetflag, tgetstr, tgoto, tputs: terminal	termcap(3)
routines. tgetent, tgetnum, tgetflag, tgetstr,	tgetnum, tgetflag, tgetstr, tgoto, tputs: terminal	termcap(3)
explain: print wordy sentences, interactive	tgetstr, tgoto, tputs: terminal independent	termcap(3)
ttt:	tgoto, tputs: terminal independent operation	termcap(3)
activity. timex:	thesaurus for diction. diction,	diction(1)
time:	tic-tac-toe.	ttt(8)
mclock: return Fortran	time a command; report process data and system	timex(1)
at: execute commands at a later	time a command.	time(1)
dcopy: copy file systems for optimal access	time accounting.	mclock(3f)
	time.	at(1)
profil: execution	time: get time.	dcopy(8)
profile: setting up an environment at login	time profile.	time(2)
stime: set	time.	profil(2)
	time: time a command.	profile(5)
time: get	time.	stime(2)
localtime, gmtime, asctime, tzset: convert date and	time: time a command.	time(1)
clock: report CPU	time to ASCII. ctime,	time(2)
	time used.	ctime(3C)
times: get process and child process	times: get process and child process times.	clock(3C)
utime: set file access and modification	times.	times(2)
system activity.	times.	times(2)
	timex: time a command; report process data and	utime(2)
ot, ox: TM 503/TM 603/TM 703 disk,	tk: paginator for the Tektronix 4014.	timex(1)
ot, ox:	TM 100-4 floppy disk.	tk(1)
	TM 503/TM 603/TM 703 disk, TM 100-4 floppy disk.	ot(4)
		ot(4)

Permuted Index

interface. tm,ts:	TM-11/TU-10 magtape interface, TS-11 magtape	tm(4)
	tmpfile: create a temporary file.	tmpfile(3S)
	tmpnam: create a name for a temporary file.	tmpnam(3S)
interface.	tm,ts: TM-11/TU-10 magtape interface, TS-11 magtape	tm(4)
toupper, tolower,	toascii: character translation.	conv(3C)
popen, pclose: initiate I/O	to/from a process.	popen(3S)
toupper,	tolower, toascii: character translation.	conv(3C)
tsort:	topological sort.	tsort(1)
acctmrg: merge or add	total accounting files.	acctmrg(8)
	touch: update date last modified of a file.	touch(1)
	toupper, tolower, toascii: character translation.	conv(3C)
	tplot: graphics filters.	tpivot(1G)
tgetent, tgetnum, tgetflag, tgetstr, tgoto,	tputs: terminal independent operation routines.	termcap(3)
	tr: translate characters.	tr(1)
ptrace: process	trace.	ptrace(2)
sign, isign, dsign: Fortran	transfer-of-sign intrinsic function.	sign(3f)
tr:	translate characters.	tr(1)
toupper, tolower, toascii: character	translation.	conv(3C)
ftw: walk a file	tree.	ftw(3C)
tsearch, tdelete, twalk: manage binary search	trees.	tsearch(3C)
sin, cos, tan, asin, acos, atan, atan2:	trigonometric functions.	trig(3M)
checkeq: format mathematical text for nroff or	troff. eqn, neqn.	eqn(1)
	troff, nroff: typeset or format text.	troff(1)
	troff.	tbi(1)
tbi: format tables for nroff or	troff to the CANON LBP or the Versatec V80.	ltroff(1)
ltroff, vtroff:	true, false: provide truth values.	true(1)
	truth values.	true(1)
true, false: provide	TS-11 magtape interface.	tm(4)
tm,ts: TM-11/TU-10 magtape interface,	tsearch, tdelete, twalk: manage binary search	tsearch(3C)
trees.	tsort: topological sort.	tsort(1)
	ttt: tic-tac-toe.	ttt(8)
	tty: controlling terminal interface.	tty(4)
	tty: get terminal name.	tty(1)
greek: graphics for the extended	TTY-37 type-box.	greek(7)
	ttname, isatty: find name of a terminal.	ttname(3C)
current user.	ttyslot: find the slot in the utmp file of the	ttyslot(3C)
prdaily, prtacct, runacct, shutacct, startup,	turnacct: shell procedures for accounting. /prctmp,	acctsh(8)
startrek: THE game based on the	t.v. series..	startrek(8)
tsearch, tdelete,	twalk: manage binary search trees.	tsearch(3C)
dbl, cmplx, demplx, ichar, char: explicit Fortran	type conversion. /iflx, idint, real, float, snl,	ftype(3f)
file: determine file	type.	file(1)
getty: set terminal	type, modes, speed, and line discipline.	getty(8)
greek: graphics for the extended TTY-37	type-box.	greek(7)
	types: primitive system data types.	types(7)
types: primitive system data	types.	types(7)
script: make	typescript of terminal session.	script(1)
mmt, mvt:	typeset documents, view graphs, and slides.	mmt(1)
troff, nroff:	typeset or format text.	troff(1)
ctime, localtime, gmtime, asctime,	tzset: convert date and time to ASCII	ctime(3C)
getpw: get name from	UID.	getpw(3C)
	ul: do underlining.	ul(1)
	ulimit: get and set user limits.	ulimit(2)
	umask: set and get file creation mask.	umask(2)
	umask: set file-creation mode mask.	umask(1)
mount,	umount: mount and dismount file system.	mount(8)
	umount: unmount a file system.	umount(2)
current UNIX system.	uname, ethname: get name/ethernet-identification of	uname(2)
	uname: print name of current UNIX.	uname(1)
cat them. compact,	uncompact, ccat: compress and uncompress files, and	compact(1)
compact, uncompact, ccat: compress and	uncompress files, and cat them.	compact(1)
ul: do	underlining.	ul(1)
unget:	undo a previous get of an SCCS file.	unget(1)
expand,	unexpand: expand tabs to spaces, and vice versa.	expand(1)
	unget: undo a previous get of an SCCS file.	unget(1)
	ungetc: push character back into input stream.	ungetc(3S)
rand, srand: Fortran	uniform random-number generator.	rand(3f)
/jrand48, srand48, seed48, lcong48: generate	uniformly distributed pseudo-random numbers.	drand48(3C)
	unify relational data base system.	unify(1)
	uniq: report repeated lines in a file.	uniq(1)
mktemp: make a	unique file name.	mktemp(3C)
system.	unite: enable a remote user to access the local	unite(8N)
	units: conversion program.	units(1)
uuto, uupick: public	UNIX-to-UNIX file copy.	uuto(1C)
link,	unlink: exercise link and unlink system calls.	link(8)
	unlink: remove directory entry.	unlink(2)
link, unlink: exercise link and	unlink system calls.	link(8)
umount:	unmount a file system.	umount(2)
pack: packs or	unpacks many files <--> one..	pack(1)
recnews: receive	unprocessed articles via mail.	recnews(1)

Permuted Index

	recnews:	receive unprocessed articles via mail.	recnews(8)
	make:	maintain, update, and regenerate groups of programs.	make(1)
	touch:	update date last modified of a file.	touch(1)
	lsearch:	linear search and update.	lsearch(3C)
	sync:	update super-block.	sync(2)
	sync:	update the super block.	sync(8)
	uptime:	show how long system has been up.	uptime(1)
	du:	summarize disk usage.	du(1)
	munchek:	check usage of mm macros and eqn delimiters.	mmchek(1)
	news:	USENET network news article, utility files.	news(5)
	Connection. /etc/pwmap, /etc/groupmap:	user and group id mappings for the Newcastle	pwwmap(5N)
	id:	print user and group IDs and names.	id(1)
	setuid, setgid:	set user and group IDs.	setuid(2)
	chfn:	change full name of user.	chfn(1)
	cuserid:	character login name of the user.	cuserid(3S)
	group/ getuid, geteuid, getgid, getegid:	get real user, effective user environment.	getuid(2)
	environ:	user environment.	environ(7)
	checknews:	check to see if user has news.	checknews(1)
	/etc/setugi:	alter user id of a UNIX server.	setugi(8N)
	finger:	user information lookup program.	finger(1)
	nfcomment:	a user interface to the notesfile system.	nfcomment(3)
	ulimit:	get and set user limits.	ulimit(2)
	logname:	login name of user.	logname(3X)
	geteuid, getgid, getegid:	get real user, effective user, real group, and effective group IDs.	getuid(2)
	su:	become super-user or another user.	su(1)
	unite:	enable a remote user to access the local system.	unite(8N)
	find the slot in the utmp file of the current user.	ttyslot:	ttyslot(3C)
	write:	write to another user.	write(1)
	mail, rmail:	send mail to users or read mail.	users(1)
	wall:	write to all users.	mail(1)
	users:	compact list of users who are on the system.	wall(8)
	fuser:	identify processes using a file or file structure.	users(1)
	mail.nc:	mail(1) using the Newcastle Connection instead of uucp..	fuser(8)
	usrv:	UNIX server for a remote client.	mail(1N)
	ustat:	get file system statistics.	usrv(8N)
	news:	USENET network news article, utility files.	ustat(2)
	mfrcv, mfrarchive, newsinput, newsoutput:	notesfile utility programs. mkknf, rmknf, nfxfmt,	news(5)
	utmp, wtmp:	time: set file access and modification times.	notes(8)
	pututline, setutent, endutent, utmpname:	access utmp and wtmp entry formats.	utime(2)
	ttyslot:	find the slot in the utmp file entry.	utmp(5)
	getutid, getutline, pututline, setutent, endutent,	utmp file entry. getutent, getutid, getutline,	getut(3C)
	uucp:	uucp installation made easy.	ttyslot(3C)
	uucp.. mail.nc:	uucp status inquiry and job control.	utmp(5)
	uusub:	monitor uucp network.	getut(3C)
	uuclean:	uucp spool directory clean-up.	uuclean(8)
	uustat:	uucp status inquiry and job control.	uucp(8)
	uucp, uuolog, uuuname:	unix to unix copy.	mail(1N)
	uucp, uuolog, uuuname:	unix to unix copy.	uusub(8)
	uuto, uuopick:	public UNDX-to-UNDX file copy.	uuclean(8)
	uuerec:	receive processed news articles via mail.	uustat(1C)
	uustat:	uucp status inquiry and job control.	uucp(8)
	uusub:	monitor uucp network.	uucp(1C)
	uuto, uuopick:	public UNDX-to-UNDX file copy.	uucp(1C)
	uuux:	unix to unix command execution.	uuto(1C)
	viroff:	troff to the CANON LBP or the Versatec V80. ltroff,	uuux(1C)
	val:	validate SCCS file.	ltroff(1)
	abs:	integer absolute value.	val(1)
	abs, iabs, dabs, cabs, zabs:	Fortran absolute value.	abs(3C)
	getenv:	value for environment name.	abs(3f)
	fmod, fabs:	floor, ceiling, remainder, absolute values.	getenv(3C)
	true, false:	provide truth variable.	floor(3M)
	getenv:	return Fortran environment vector and device addresses.	true(1)
	configuration information:	table of interrupt verifier.	getenv(3f)
	lint:	a C program verify program assertion.	vc(1)
	assert:	versa.	confinfo(4)
	expand, unexpand:	expand tabs to spaces, and vice versa.	lint(1)
	vp:	Versatec printer-plotter.	assert(3X)
	ltroff, vtroff:	troff to the CANON LBP or the Versatec V80.	expand(1)
	vfont:	font formats for the Benson-Varian or Versatec.	vp(4)
	vc:	version control.	ltroff(1)
	get:	get a version of an SCCS file.	vfont(5)
			vc(1)
			get(1)

Permuted Index

scsdiff: compare two	versions of an SCCS file.	scsdiff(1)
Versatec.	font formats for the Benson-Varian or	vfont(5)
font.	vfontinfo: inspect and print out information about	vfontinfo(1)
on ex.	vi: screen oriented (visual) display editor based	vi(1)
expand, unexpand: expand tabs to spaces, and	vice versa.	expand(1)
mmt, mvt: typeset documents.	view graphs, and slides.	mmt(1)
mv: a macro package for making	view graphs.	mv(7)
more, page: file perusal filter for crt	viewing.	more(1)
vi: screen oriented	(visual) display editor based on ex.	vi(1)
checking.	voicopy, labelit: copy file systems with label	voicopy(8)
fs: file system format of system	volume.	fs(5)
	vp: Versatec printer-plotter.	vp(4)
	vtroff: troff to the CANON LBP or the Versatec V80.	ltroff(1)
ltroff,	w: who is on and what they are doing.	w(1)
	wait: await completion of process.	wait(1)
wait:	wait for child process to stop or terminate.	wait(2)
	wait: wait for child process to stop or terminate.	wait(2)
ftw:	walk a file tree.	ftw(3C)
	wall: write to all users.	wall(8)
	wc: word count.	wc(1)
whatis: describe	what a command is.	whatis(1)
whatconf:	what device drivers are in an unix kernel.	whatconf(8)
	what: identify SCCS files.	what(1)
w: who is on and	what they are doing.	w(1)
signal: specify	what to do upon receipt of a signal.	signal(2)
crash-	what to do when the system crashes.	crash(8)
kernel.	whatconf: what device drivers are in an unix	whatconf(8)
	whatis: describe what a command is.	whatis(1)
program.	whereis: locate source, binary, and or manual for	whereis(1)
users: compact list of users	who are on the system.	users(1)
whodo:	who is doing what.	whodo(8)
from:	who is my mail from?.	from(1)
w:	who is on and what they are doing.	w(1)
who:	who is on the system.	who(1)
	who: who is on the system.	who(1)
fold: fold long lines for finite	whodo: who is doing what.	whodo(8)
diction, explain: print	width output device.	fold(1)
cd: change	wordy sentences, interactive thesaurus for diction.	diction(1)
chdir: change	working directory.	cd(1)
getcwd: get path-name of current	working directory.	chdir(2)
pwd:	working directory.	getcwd(3C)
write:	working directory name.	pwd(1)
putpwent:	write on a file.	write(2)
wall:	write password file entry.	putpwent(3C)
write:	write to all users.	wall(8)
	write: write to another user.	write(1)
	write: write on a file.	write(2)
	write: write to another user.	write(1)
open: open for reading or	writing.	open(2)
utmp, wtmp: utmp and	wtmp entry formats.	utmp(5)
utmp,	wtmp: utmp and wtmp entry formats.	utmp(5)
ftwtmp,	wtmpfix: manipulate connect accounting records.	ftwtmp(3)
	wump: the game of hunt-the-wumpus.	wump(8)
command.	xargs: construct argument list(s) and execute	xargs(1)
	xd: hexadecimal dump.	xd(1)
functions. and, or,	xor, not, lshift, rshift: Fortran bitwise boolean	bool(3)
	xref: cross-reference listing.	xref(1)
shared strings.	xstr: extract strings from C programs to implement	xstr(1)
j0, j1, jn,	y0, y1, yn: Bessel functions.	bessel(3M)
j0, j1, jn, y0,	y1, yn: Bessel functions.	bessel(3M)
	yacc: yet another compiler-compiler.	yacc(1)
j0, j1, jn, y0, y1,	yes: be repetitively affirmative.	yes(1)
abs, iabs, dabs, cabs,	yn: Bessel functions.	bessel(3M)
	zabs: Fortran absolute value.	abs(3f)

NAME

a68 - MIT assembler

SYNOPSIS

a68 [-o objectfile] [-alu] sourcefile

a68 file.s writes object 'file.o', or the '-o objectfile'.

a68 -a writes all symbols, not just globals, in 'file.o'.

a68 -l writes a listing on stdout, e.g.
a68 -l file.s >file.lis

a68 -u makes undefined symbols .extern, instead of errors.

DESCRIPTION

A68 is the MIT Trix assembler; see /usr/doc/a68. A68 is NOT compatible with AS(1); it has no macros, and wants e.g. 'movw' instead of 'MOVE' or 'MOVE.W'. (There is a filter to change Motorola assembler format to A68, and another to expand macros).

SEE ALSO

/usr/doc/a68

NAME

DATE

STATION

TIME

WIND

SEA

TEMP

WIND

SEA

WIND

SEA

WIND

SEA

WIND

SEA

NAME

acctcom - search and print process accounting file(s)

SYNOPSIS

acctcom [[options][file]] . . .

DESCRIPTION

Acctcom reads *file*, the standard input, or */usr/adm/pacct*, in the form described by *acct(5)* and writes selected records to the standard output. Each record represents the execution of one process. The output shows the **COMMAND NAME**, **USER**, **TTYNAME**, **START TIME**, **END TIME**, **REAL (SEC)**, **CPU (SEC)**, **MEAN SIZE(K)**, and optionally, **F** (the *fork/exec* flag: 1 for *fork* without *exec*) and **STAT** (the system exit status).

The command name is prepended with a # if it was executed with *super-user* privileges. If a process is not associated with a known terminal, a ? is printed in the **TTYNAME** field.

If no *files* are specified, and if the standard input is associated with a terminal or */dev/null* (as is the case when using *&* in the shell), */usr/adm/pacct* is read, otherwise the standard input is read.

If any *file* arguments are given, they are read in their respective order. Each file is normally read forward, i.e., in chronological order by process completion time. The file */usr/adm/pacct* is usually the current file to be examined; a busy system may need several such files of which all but the current file are found in */usr/adm/pacct?*. The *options* are:

- b Read backwards, showing latest commands first.
- f Print the *fork/exec* flag and system exit status columns in the output.
- h Instead of mean memory size, show the fraction of total available CPU time consumed by the process during its execution. This "hog factor" is computed as:
(total CPU time)/(elapsed time).
- i Print columns containing the I/O counts in the output.
- k Instead of memory size, show total kcore-minutes.
- m Show mean core size (the default).
- r Show CPU factor (user time/(system-time + user-time)).
- t Show separate system and user CPU times.
- v Exclude column headings from the output.
- l *line* Show only processes belonging to terminal */dev/line*.
- u *user* Show only processes belonging to *user* that may be specified by: a user ID, a login name that is then converted to a user ID, a # which designates only those processes executed with *super-user* privileges, or ? which designates only those processes associated with unknown user IDs.
- g *group* Show only processes belonging to *group*. The *group* may be designated by either the group ID or group name.
- d *mm/dd* Any *time* arguments following this flag are assumed to occur on the given month *mm* and the day *dd* rather than during last 24 hours. This is needed for looking at old files.
- s *time* Select processes existing at or after *time*, given in the format *hr[:min[:sec]]*.

- e *time* Select processes existing at or before *time*.
- S *time* Select processes starting at or after *time*.
- E *time* Select processes ending at or before *time*.
- n *pattern* Show only commands matching *pattern* that may be a regular expression as in *ed*(1) except that + means one or more occurrences.
- o *ofile* Copy selected process records in the input data format to *ofile*; suppress standard output printing.
- H *factor* Show only processes that exceed *factor*, where *factor* is the "hog factor" as explained in option -h above.
- O *sec* Show only processes with CPU system time exceeding *sec* seconds.
- C *sec* Show only processes with total CPU time, system plus user, exceeding *sec* seconds.

Listing options together has the effect of a logical *and*.

FILES

/etc/passwd
/usr/adm/pacct
/etc/group

SEE ALSO

ps(1), *su*(1), *acct*(2), *acct*(5), *utmp*(5).
acct(8), *acctcms*(8), *acctcon*(8), *acctmerg*(8), *acctprc*(8), *acctsh*(8),
fwtmp(8), *runacct*(8).

BUGS

Acctcom only reports on processes that have terminated; use *ps*(1) for active processes. If *time* exceeds the present time and option -d is not used, then *time* is interpreted as occurring on the previous day.

NAME

adb - debugger

SYNOPSIS

adb [-w] [*objfil* [*corfil*]]

DESCRIPTION

Adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of UNIX programs.

Objfil is normally an executable program file, preferably containing a symbol table; if not then the symbolic features of *adb* cannot be used although the file can still be examined. The default for *objfil* is *a.out*. *Corfil* is assumed to be a core image file produced after executing *objfil*; the default for *corfil* is *core*.

Requests to *adb* are read from the standard input and responses are to the standard output. If the -w flag is present then both *objfil* and *corfil* are created if necessary and opened for reading and writing so that files can be modified using *adb*. *Adb* ignores QUIT; INTERRUPT causes return to the next *adb* command.

In general requests to *adb* are of the form

[*address*] [, *count*] [*command*] [;]

If *address* is present then *dot* is set to *address*. Initially *dot* is set to 0. For most commands *count* specifies how many times the command will be executed. The default *count* is 1. *Address* and *count* are expressions.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged then addresses are interpreted in the usual way in the address space of the subprocess. For further details of address mapping see ADDRESSES.

EXPRESSIONS

- .
 - +
 - ~
 - "
- The value of *dot*.
The value of *dot* incremented by the current increment.
The value of *dot* decremented by the current increment.
The last *address* typed.

integer

An octal number if *integer* begins with a 0; a hexadecimal number if preceded by #; otherwise a decimal number.

integer.fraction

A 32 bit floating point number.

'cccc' The ASCII value of up to 4 characters. \ may be used to escape a

< *name*

The value of *name*, which is either a variable name or a register name. *Adb* maintains a number of variables (see VARIABLES) named by single letters or digits. If *name* is a register name then the value of the register is obtained from the system header in *corfil*. The register names are d0 ... d7 a0 ... a7 pc ns ac ip. The register

names *ns*, *ac* and *ip* deserve special explanation. The register *ns* is the concatenation of the 2 byte exception vector number plus the 2 byte 68000 *cpustate* stored during bus or address error exceptions. If e.g. the first two bytes of *ns* are 0002, the program was interrupted by a bus error (exception vector 2). The register *ac* contains the access address for a bus or address error exception. The register *ip* is the concatenation of the 2 byte instruction register, i.e. the first two bytes of the instruction that caused a bus or address error, and the 2 byte processor status. For exceptions that are not bus or address error, the second half of *ns*, all of *ac*, and the first half of *ip* are 0.

symbol

A *symbol* is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. The value of the *symbol* is taken from the symbol table in *objfil*. An initial `_` or `~` will be prepended to *symbol* if needed.

`_symbol`

In C, the 'true name' of an external symbol begins with `_`. It may be necessary to utter this name to distinguish it from internal or hidden variables of a program.

routine.name

The address of the variable *name* in the specified C routine. Both *routine* and *name* are *symbols*. If *name* is omitted the value is the address of the most recently activated C stack frame corresponding to *routine*.

(*exp*) The value of the expression *exp*.

Monadic operators

`*exp` The contents of the location addressed by *exp* in *corfil*.

`@exp` The contents of the location addressed by *exp* in *objfil*.

`-exp` Integer negation.

`~exp` Bitwise complement.

Dyadic operators are left associative and are less binding than monadic operators.

e1+*e2* Integer addition.

e1-*e2* Integer subtraction.

*e1***e2* Integer multiplication.

e1%*e2* Integer division.

e1&*e2* Bitwise conjunction.

e1|*e2* Bitwise disjunction.

e1#*e2* *E1* rounded up to the next multiple of *e2*.

COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands '?' and '/' may be followed by '*'; see ADDRESSES for further details.)

- ?f Locations starting at *address* in *objfil* are printed according to the format *f*.
- /f Locations starting at *address* in *corfil* are printed according to the format *f*.
- =f The value of *address* itself is printed in the styles indicated by the format *f*. (For i format '?' is printed for the parts of the instruction that reference subsequent words.)

A *format* consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format *dot* is incremented temporarily by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows.

- o 2 Print 2 bytes in octal. All octal numbers output by *adb* are preceded by 0.
- O 4 Print 4 bytes in octal.
- q 2 Print in signed octal.
- Q 4 Print long signed octal.
- d 2 Print in decimal.
- D 4 Print long decimal.
- x 2 Print 2 bytes in hexadecimal.
- X 4 Print 4 bytes in hexadecimal.
- u 2 Print as an unsigned decimal number.
- U 4 Print long unsigned decimal.
- f 4 Print the 32 bit value as a floating point number. Currently only FFP format supported (see *fp(3)*).
- F 8 Print double floating point. For FFP same as above, except larger printing format.
- b 1 Print the addressed byte in octal.
- c 1 Print the addressed character.
- C 1 Print the addressed character using the following escape convention. Character values 000 to 040 are printed as @ followed by the corresponding character in the range 0100 to 0140. The character @ is printed as @@.
- s n Print the addressed characters until a zero character is reached.
- S n Print a string using the @ escape convention. *n* is the length of the string including its zero terminator.
- Y 4 Print 4 bytes in date format (see *ctime(3C)*).
- i n Print as 68000 instructions. *n* is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.
- a 0 Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.
 - / local or global data symbol
 - ? local or global text symbol
 - = local or global absolute symbol

- p 2** Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.
- t 0** When preceded by an integer tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.
- r 0** Print a space.
- n 0** Print a newline.
- "..." 0** Print the enclosed string.
- ~** *Dot* is decremented by the current increment. Nothing is printed.
- +** *Dot* is incremented by 1. Nothing is printed.
- *Dot* is decremented by 1. Nothing is printed.

newline

If the previous command temporarily incremented *dot*, make the increment permanent. Repeat the previous command with a *count* of 1.

[?/]l value mask

Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If **L** is used then the match is for 4 bytes at a time instead of 2. If no match is found then *dot* is unchanged; otherwise *dot* is set to the matched location. If *mask* is omitted then -1 is used.

[?/]w value ...

Write the 2-byte *value* into the addressed location. If the command is **W**, write 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

[?/]m b1 e1 f1[?/]

New values for (*b1*, *e1*, *f1*) are recorded. If less than three expressions are given then the remaining map parameters are left unchanged. If the '?' or '/' is followed by '*' then the second segment (*b2*, *e2*, *f2*) of the mapping is changed. If the list is terminated by '?' or '/' then the file (*objfil* or *corfil* respectively) is used for subsequent requests. (So that, for example, '/m?' will cause '/' to refer to *objfil*.)

>name

Dot is assigned to the variable or register named.

! A shell is called to read the rest of the line following '!'.

\$modifier

Miscellaneous commands. The available *modifiers* are:

- <f** Read commands from the file *f* and return.
- >f** Send output to the file *f*, which is created if it does not exist.
- r** Print the general registers and the instruction addressed by *pc*. *Dot* is set to *pc*.
- b** Print all breakpoints and their associated counts and commands.
- c** C stack backtrace. If *address* is given then it is taken as the address of the current frame (instead of *a6*). If **C** is used then the names and (32 bit) values of all automatic

and static variables are printed for each active function. If *count* is given then only the first *count* frames are printed. If the module was compiled with the -L option of cc(1), the procedure name is followed by the current linenumber. For all variables 32 bits are printed, even if the variable is only char or short. In this case only the foremost bytes are to be considered. For register variables, however, the whole register in which the variable resides is printed, and so for shorts the last two bytes are the relevant ones! In order to help you make the distinction between register and normal variables, the latter are prefixed with a "<".

- e The names and values of external variables are printed.
- w Set the page width for output to *address* (default 80).
- s Set the limit for symbol matches to *address* (default 255).
- o All integers input are regarded as octal.
- d Reset integer input as described in EXPRESSIONS.
- q Exit from *adb*.
- v Print all non zero variables in octal.
- m Print the address map.

:modifier

Manage a subprocess. Available modifiers are:

- bc Set breakpoint at *address*. The breakpoint is executed *count*-1 times before causing a stop. Each time the breakpoint is encountered the command *c* is executed. If this command sets *dot* to zero then the breakpoint causes a stop. If the module has been compiled with option -g, see cc(1), you can set a breakpoint at line 75 by saying L75:b.
- d Delete breakpoint at *address*.
- r Run *objfil* as a subprocess. If *address* is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on entry to the subprocess.
- cs The subprocess is continued with signal *s*, see *signal(2)*. If *address* is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for *r*.
- ss As for *c* except that the subprocess is single stepped *count* times. If there is no current subprocess then *objfil* is run as a subprocess as for *r*. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.
- k The current subprocess, if any, is terminated.

VARIABLES

Adb provides a number of variables. Named variables are set initially by *adb* but are not used subsequently. Numbered variables are reserved for communication as follows.

- 0 The last value printed.
- 1 The last offset part of an instruction source.
- 2 The previous value of variable 1.

On entry the following are set from the system header in the *corfil*. If *corfil* does not appear to be a core file then these values are set from *objfil*.

- b* The base address of the data segment.
- d* The data segment size.
- e* The entry point.
- m* The 'magic' number (0405, 0407, 0410 or 0411).
- s* The stack segment size.
- t* The text segment size.

ADDRESSES

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (*b1*, *e1*, *f1*) and (*b2*, *e2*, *f2*) and the *file address* corresponding to a written *address* is calculated as follows.

$b1 \leq \text{address} < e1 \Rightarrow \text{file address} = \text{address} + f1 - b1$, otherwise,

$b2 \leq \text{address} < e2 \Rightarrow \text{file address} = \text{address} + f2 - b2$,

otherwise, the requested *address* is not legal. In some cases (e.g. for programs with separated I and D space) the two segments for a file may overlap. If a ? or / is followed by an * then only the second triple is used.

The initial setting of both mappings is suitable for normal *a.out* and core files. If either file is not of the kind expected then, for that file, *b1* is set to 0, *e1* is set to the maximum file size and *f1* is set to 0; in this way the whole file can be examined with no address translation.

So that *adb* may be used on large files all appropriate values are kept as signed 32 bit integers.

FILES

/dev/mem
/dev/swap
a.out
core

SEE ALSO

ptrace(2), a.out(5), core(5)

DIAGNOSTICS

'Adb' when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or returned nonzero status.

BUGS

A breakpoint set at the entry point is not effective on initial entry to the program.

When single stepping, system calls do not count as an executed

instruction.

Local variables whose names are the same as an external variable may foul up the accessing of the external.

EXAMPLES

After a core dump:

```
adb prog      (second parameter core is assumed)
$r           (print registers)
$c           (print C stack trace)
$C           (print full C stack trace)
$e           (print external variables)
bufp/X       (print contents of bufp as long hex)
*bufp/X      (print 4 bytes hex where bufp points to)
write?i      (prints first instruction, link a6,... )
(cr)         (print next instruction)
:b           (set breakpoint at this instruction)
:r hello <inp (run prog with param hello and input from inp)
main,-1?ia   (disassemble program starting at main)
data/4X2d    (print data: 4 long hex, 2 short decimal)
(cr)         (print next data in same format)
#100>d0      (write #100 to register d0)
<d1=X        (print d1 long hex)
*( <a6+8)/XXX (print first three longs of parameter area)
write?l #4e75 (search for rts instruction at end of proc write)
:b           (set breakpoint at this address)
:c           (continue program until stop or breakpoint)
:s           (single step through program)
```

etc.

On the 68000, as opposed to the 68010 or 68020, the stack will not necessarily grow automatically. You can recognize a stackoverflow with the following commands:

```
adb prog
$r
$m
```

If the value of register ac is less than the value b2 for the second (/) map, the stack overflowed. To get the needed stack size, type

```
#f00000-<ac=D
```

This will output the difference between the top of the stack and the access address in decimal. Round the value up some kbyte, and give the command

```
stksiz prog <value>
```


...the ... of ...

...the ... of ...

...the ... of ...

...the ... of ...

...the ... of ...

...the ... of ...

NAME

`admin` - create and administer SCCS files

SYNOPSIS

```
admin [-n] [-i[name]] [-rrel] [-t[name]] [-fflag[flag-val]]
      [-dflag[flag-val]] [-alogin] [-eloin] [-m[mrlist]] [-y[comment]] [-h]
      [-z] files
```

DESCRIPTION

Admin is used to create new SCCS files and change parameters of existing ones. Arguments to *admin*, which may appear in any order, consist of keyletter arguments, which begin with `-`, and named files (note that SCCS file names must begin with the characters `s.`). If a named file doesn't exist, it is created, and its parameters are initialized according to the specified keyletter arguments. Parameters not initialized by a keyletter argument are assigned a default value. If a named file does exist, parameters corresponding to specified keyletter arguments are changed, and other parameters are left as is.

If a directory is named, *admin* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with `s.`) and unreadable files are silently ignored. If a name of `-` is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed since the effects of the arguments apply independently to each named file.

`-n` This keyletter indicates that a new SCCS file is to be created.

`-i[name]` The *name* of a file from which the text for a new SCCS file is to be taken. The text constitutes the first delta of the file (see `-r` keyletter for delta numbering scheme). If the `i` keyletter is used, but the file name is omitted, the text is obtained by reading the standard input until an end-of-file is encountered. If this keyletter is omitted, then the SCCS file is created empty. Only one SCCS file may be created by an *admin* command on which the `i` keyletter is supplied. Using a single *admin* to create two or more SCCS files require that they be created empty (no `-i` keyletter). Note that the `-i` keyletter implies the `-n` keyletter.

`-rrel` The *release* into which the initial delta is inserted. This keyletter may be used only if the `-i` keyletter is also used. If the `-r` keyletter is not used, the initial delta is inserted into release 1. The level of the initial delta is always 1 (by default initial deltas are named 1.1).

`-t[name]` The *name* of a file from which descriptive text for the SCCS file is to be taken. If the `-t` keyletter is used and *admin* is creating a new SCCS file (the `-n` and/or `-i`

keyletters also used), the descriptive text file name must also be supplied. In the case of existing SCCS files: (1) a `-t` keyletter without a file name causes removal of descriptive text (if any) currently in the SCCS file, and (2) a `-t` keyletter with a file name causes text (if any) in the named file to replace the descriptive text (if any) currently in the SCCS file.

`-f flag`

This keyletter specifies a *flag*, and, possibly, a value for the *flag*, to be placed in the SCCS file. Several keyletters may be supplied on a single *admin* command line. The allowable *flags* and their values are:

- b** Allows use of the `-b` keyletter on a *get(1)* command to create branch deltas.
- cceil** The highest release (i.e., "ceiling"), a number less than or equal to 9999, which may be retrieved by a *get(1)* command for editing. The default value for an unspecified *c* flag is 9999.
- ffloor** The lowest release (i.e., "floor"), a number greater than 0 but less than 9999, which may be retrieved by a *get(1)* command for editing. The default value for an unspecified *f* flag is 1.
- dSID** The default delta number (SID) to be used by a *get(1)* command.
- i** Causes the "No id keywords (ge6)" message issued by *get(1)* or *delta(1)* to be treated as a fatal error. In the absence of this flag, the message is only a warning. The message is issued if no SCCS identification keywords (see *get(1)*) are found in the text retrieved or stored in the SCCS file.
- j** Allows concurrent *get(1)* commands for editing on the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.
- list** A *list* of releases to which deltas can no longer be made (*get -e* against one of these "locked" releases fails). The *list* has the following syntax:

```
<list> ::= <range> | <list> , <range>
<range> ::= RELEASE NUMBER | a
```

The character *a* in the *list* is equivalent to specifying *all releases* for the named SCCS file.
- n** Causes *delta(1)* to create a "null" delta in each of those releases (if any) being skipped when a delta is made in a *new* release (e.g., in making delta 5.1 after delta 2.7, releases 3 and 4 are skipped). These null deltas serve as "anchor points" so that branch deltas may later be created from them. The absence of this flag causes skipped releases to be non-existent in the SCCS file preventing branch deltas from being created

from them in the future.

qtext User definable text substituted for all occurrences of the %Q% keyword in SCCS file text retrieved by *get(1)*.

mmod Module name of the SCCS file substituted for all occurrences of the %M% keyword in SCCS file text retrieved by *get(1)*. If the **m** flag is not specified, the value assigned is the name of the SCCS file with the leading **s.** removed.

ttype Type of module in the SCCS file substituted for all occurrences of %Y% keyword in SCCS file text retrieved by *get(1)*.

v[pgm]

Causes *delta(1)* to prompt for Modification Request (MR) numbers as the reason for creating a delta. The optional value specifies the name of an MR number validity checking program (see *delta(1)*). (If this flag is set when creating an SCCS file, the **m** keyletter must also be used even if its value is null).

-dflag

Causes removal (deletion) of the specified *flag* from an SCCS file. The **-d** keyletter may be specified only when processing existing SCCS files. Several **-d** keyletters may be supplied on a single *admin* command. See the **-f** keyletter for allowable *flag* names.

llist

A list of releases to be "unlocked". See the **-f** keyletter for a description of the **l** flag and the syntax of a list.

-alogin

A *login* name, or numerical UNIX group ID, to be added to the list of users which may make deltas (changes) to the SCCS file. A group ID is equivalent to specifying all *login* names common to that group ID. Several *login* names may be used on a single *admin* command line. As many *logins*, or numerical group IDs, as desired may be on the list simultaneously. If the list of users is empty, then anyone may add deltas.

-ellogin

A *login* name, or numerical group ID, to be erased from the list of users allowed to make deltas (changes) to the SCCS file. Specifying a group ID is equivalent to specifying all *login* names common to that group ID. Several *e* keyletters may be used on a single *admin* command line.

-y[comment]

The *comment* text is inserted into the SCCS file as a comment for the initial delta in a manner identical to that of *delta(1)*. Omission of the **-y** keyletter results in a default comment line being inserted in the form:

date and time created YY/MM/DD HH:MM:SS by *login*

The **-y** keyletter is valid only if the **-i** and/or **-n** keyletters are specified (i.e., a new SCCS file is being

created).

-m[mrlist]

The list of Modification Requests (*MR*) numbers is inserted into the SCCS file as the reason for creating the initial delta in a manner identical to *delta(1)*. The *v* flag must be set and the *MR* numbers are validated if the *v* flag has a value (the name of an *MR* number validation program). Diagnostics will occur if the *v* flag is not set or *MR* validation fails.

-h

Causes *admin* to check the structure of the SCCS file (see *sccsfile(5)*), and to compare a newly computed check-sum (the sum of all the characters in the SCCS file except those in the first line) with the check-sum that is stored in the first line of the SCCS file. Appropriate error diagnostics are produced.

This keyletter inhibits writing on the file, so that it nullifies the effect of any other keyletters supplied, and is, therefore, only meaningful when processing existing files.

-z

The SCCS file check-sum is recomputed and stored in the first line of the SCCS file (see **-h**, above).

Note that use of this keyletter on a truly corrupted file may prevent future detection of the corruption.

FILES

The last component of all SCCS file names must be of the form *s.file-name*. New SCCS files are given mode 444 (see *chmod(1)*). Write permission in the pertinent directory is, of course, required to create a file. All writing done by *admin* is to a temporary x-file, called *x.file-name*, (see *get(1)*), created with mode 444 if the *admin* command is creating a new SCCS file, or with the same mode as the SCCS file if it exists. After successful execution of *admin*, the SCCS file is removed (if it exists), and the x-file is renamed with the name of the SCCS file. This ensures that changes are made to the SCCS file only if no errors occurred.

It is recommended that directories containing SCCS files be mode 755 and that SCCS files themselves be mode 444. The mode of the directories allows only the owner to modify SCCS files contained in the directories. The mode of the SCCS files prevents any modification at all except by SCCS commands.

If it should be necessary to patch an SCCS file for any reason, the mode may be changed to 644 by the owner allowing use of *ed(1)*. *Care must be taken!* The edited file should *always* be processed by an *admin -h* to check for corruption followed by an *admin -z* to generate a proper check-sum. Another *admin -h* is recommended to ensure the SCCS file is valid.

Admin also makes use of a transient lock file (called *z.file-name*), which is used to prevent simultaneous updates to the SCCS file by different users. See *get(1)* for further information.

SEE ALSO

delta(1), *ed(1)*, *get(1)*, *help(1)*, *prs(1)*, *what(1)*, *sccsfile(5)*.

ADMIN(1)

MUNIX

ADMIN(1)

Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1) for explanations.

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

NAME

apropos - locate commands by keyword lookup

SYNOPSIS

apropos keyword ...

DESCRIPTION

Apropos shows which manual sections contain instances of any of the given keywords in their title. Each word is considered separately and case of letters is ignored. Words which are part of other words are considered thus looking for *compile* will hit all instances of *'compiler'* also.

Try

apropos password

and

apropos editor

If the line starts *'name(section) ...'* you can do *'man section name'* to get the documentation for it. Try *'apropos format'* and then *'man 3s printf'* to get the manual on the subroutine *printf*.

Apropos is actually just the *-k* option to the *man(1)* command.

FILES

/usr/lib/whatis

data base

SEE ALSO

man(1), *whatis*(1), *catman*(8)

AUTHOR

William Joy

NAME

ar - archive and library maintainer

SYNOPSIS

ar key [posname] afile name ...

DESCRIPTION

Ar maintains groups of files combined into a single archive file. Its main use is to create and update library files as used by the link editor. It can be used, though, for any similar purpose.

Key is one character from the set *drqtpmx*, optionally concatenated with one or more of *vuaibcl*. *Afile* is the archive file. The *names* are constituent files in the archive file. The meanings of the *key* characters are:

- d** Delete the named files from the archive file.
- r** Replace the named files in the archive file. If the optional character **u** is used with **r**, then only those files with modified dates later than the archive files are replaced. If an optional positioning character from the set **abi** is used, then the *posname* argument must be present and specifies that new files are to be placed after (**a**) or before (**b** or **i**) *posname*. Otherwise new files are placed at the end.
- q** Quickly append the named files to the end of the archive file. Optional positioning characters are invalid. The command does not check whether the added members are already in the archive. Useful only to avoid quadratic behavior when creating a large archive piece-by-piece.
- t** Print a table of contents of the archive file. If no names are given, all files in the archive are tabled. If names are given, only those files are tabled.
- p** Print the named files in the archive.
- m** Move the named files to the end of the archive. If a positioning character is present, then the *posname* argument must be present and, as in **r**, specifies where the files are to be moved.
- x** Extract the named files. If no names are given, all files in the archive are extracted. In neither case does **x** alter the archive file.
- v** Verbose. Under the verbose option, *ar* gives a file-by-file description of the making of a new archive file from the old archive and the constituent files. When used with **t**, it gives a long listing of all information about the files. When used with **x**, it precedes each file with a name.
- c** Create. Normally *ar* will create *afile* when it needs to. The create option suppresses the normal message that is produced when *afile* is created.
- l** Local. Normally *ar* places its temporary files in the directory */tmp*. This option causes them to be placed in the local directory.

FILES

*/tmp/v** temporaries

SEE ALSO

ranlib(1), ld(1), lorder(1), ar(5).

BUGS

If the same file is mentioned twice in an argument list, it may be put in the archive twice.

NAME

as - assembler

SYNOPSIS

as file ...

DESCRIPTION

As assembles the named files, which must have the suffix '.s'. The output of the assembly is left on the corresponding files with suffix '.o'. Assembler listings are produced on the corresponding files with suffix '.lst'.

FILES

/lib/cpp	C preprocessor
/lib/asm	assembler
/usr/tmp/asm*	temporary files
/lib/symfile	used for initialization of /lib/asm
name.s	assembler source
name.o	assembled module
name.lst	assembler listing

SEE ALSO

Ulrike Weng-Beckmann, *Assembler 68000 Users Guide*
Motorola, *Macro Assembler Reference Manual*

DIAGNOSTICS

Diagnostics are given in the assembler listing.

1. The first part of the report is a general introduction to the subject of the study. It discusses the importance of the study and the objectives of the research.

2. The second part of the report is a detailed description of the methodology used in the study. It includes information about the sample size, the data collection methods, and the statistical analysis techniques.

3. The third part of the report is a discussion of the results of the study. It presents the findings of the research and compares them with the existing literature.

4. The fourth part of the report is a conclusion and a list of references. It summarizes the main findings of the study and provides a list of the sources used in the research.

NAME

at - execute commands at a later time

SYNOPSIS

at time [day] [file]

DESCRIPTION

At squirrels away a copy of the named *file* (standard input default) to be used as input to *sh*(1) at a specified later time. A *cd*(1) command to the current directory is inserted at the beginning, followed by assignments to all environment variables. When the script is run, it uses the user and group ID of the creator of the copy file.

The *time* is 1 to 4 digits, with an optional following 'A', 'P', 'N' or 'M' for AM, PM, noon or midnight. One and two digit numbers are taken to be hours, three and four digits to be hours and minutes. If no letters follow the digits, a 24 hour clock time is understood.

The optional *day* is either (1) a month name followed by a day number, or (2) a day of the week; if the word 'week' follows invocation is moved seven days further off. Names of months and days may be recognizably truncated. Examples of legitimate commands are

at 8am jan 24
at 1530 fr week

At programs are executed by periodic execution of the command */usr/lib/atrun* from *cron*(8). The granularity of *at* depends upon how often *atrun* is executed.

Standard output or error output is lost unless redirected.

FILES

/usr/spool/at/yy.ddd.hhhh.uu
activity to be performed at hour *hhhh* of year day *ddd* of year *yy*. *uu* is a unique number.
/usr/spool/at/lasttimedone contains *hhhh* for last hour of activity.
/usr/spool/at/past directory of activities now in progress
/usr/lib/atrun program that executes activities that are due
pwd(1)

SEE ALSO

calendar(1), *cron*(8)

DIAGNOSTICS

Complains about various syntax errors and times out of range.

BUGS

Due to the granularity of the execution of */usr/lib/atrun*, there may be bugs in scheduling things almost exactly 24 hours into the future.

The first part of the report deals with the general situation of the country and the progress of the work. It is followed by a detailed account of the work done during the year, and a summary of the results. The report is divided into two main parts, the first of which deals with the general situation of the country and the progress of the work, and the second with the detailed account of the work done during the year and the summary of the results.

The second part of the report deals with the detailed account of the work done during the year and the summary of the results. It is divided into two main parts, the first of which deals with the general situation of the country and the progress of the work, and the second with the detailed account of the work done during the year and the summary of the results.

The third part of the report deals with the detailed account of the work done during the year and the summary of the results. It is divided into two main parts, the first of which deals with the general situation of the country and the progress of the work, and the second with the detailed account of the work done during the year and the summary of the results.

The fourth part of the report deals with the detailed account of the work done during the year and the summary of the results. It is divided into two main parts, the first of which deals with the general situation of the country and the progress of the work, and the second with the detailed account of the work done during the year and the summary of the results.

The fifth part of the report deals with the detailed account of the work done during the year and the summary of the results. It is divided into two main parts, the first of which deals with the general situation of the country and the progress of the work, and the second with the detailed account of the work done during the year and the summary of the results.

The sixth part of the report deals with the detailed account of the work done during the year and the summary of the results. It is divided into two main parts, the first of which deals with the general situation of the country and the progress of the work, and the second with the detailed account of the work done during the year and the summary of the results.

NAME

awk - pattern scanning and processing language

SYNOPSIS

awk [-Fc] [prog] [file] ...

DESCRIPTION

Awk scans each input *file* for lines that match any of a set of patterns specified in *prog*. With each pattern in *prog* there can be an associated action that will be performed when a line of a *file* matches the pattern. The set of patterns may appear literally as *prog*, or in a file specified as *-f file*.

Files are read in order; if there are no files, the standard input is read. The file name '-' means the standard input. Each line is matched against the pattern portion of every pattern-action statement; the associated action is performed for each matched pattern.

An input line is made up of fields separated by white space. (This default can be changed by using FS, *vide infra*.) The fields are denoted \$1, \$2, ... ; \$0 refers to the entire line.

A pattern-action statement has the form

```
pattern { action }
```

A missing { action } means print the line; a missing pattern always matches.

An action is a sequence of statements. A statement can be one of the following:

```
if ( conditional ) statement [ else statement ]
while ( conditional ) statement
for ( expression ; conditional ; expression ) statement
break
continue
{ [ statement ] ... }
variable = expression
print [ expression-list ] [ >expression ]
printf format [ , expression-list ] [ >expression ]
next # skip remaining patterns on this input line
exit # skip the rest of the input
```

Statements are terminated by semicolons, newlines or right braces. An empty expression-list stands for the whole line. Expressions take on string or numeric values as appropriate, and are built using the operators +, -, *, /, %, and concatenation (indicated by a blank). The C operators ++, --, +=, -=, *=, /=, and %= are also available in expressions. Variables may be scalars, array elements (denoted x[i]) or fields. Variables are initialized to the null string. Array subscripts may be any string, not necessarily numeric; this allows for a form of associative memory. String constants are quoted "...".

The *print* statement prints its arguments on the standard output (or on a file if *>file* is present), separated by the current output field separator, and terminated by the output record separator. The *printf* statement formats its expression list according to the format (see *printf(3S)*).

The built-in function *length* returns the length of its argument taken as a string, or of the whole line if no argument. There are also built-in functions *exp*, *log*, *sqr*, and *int*. The last truncates its argument to an integer. *substr(s, m, n)* returns the *n*-character substring of *s* that begins at position *m*. The function *sprintf(fmt, expr, expr, ...)* formats the expressions according to the *printf*(3S) format given by *fmt* and returns the resulting string.

Patterns are arbitrary Boolean combinations (!, ||, &&, and parentheses) of regular expressions and relational expressions. Regular expressions must be surrounded by slashes and are as in *egrep*. Isolated regular expressions in a pattern apply to the entire line. Regular expressions may also occur in relational expressions.

A pattern may consist of two patterns separated by a comma; in this case, the action is performed for all lines between an occurrence of the first pattern and the next occurrence of the second.

A relational expression is one of the following:

```
expression matchop regular-expression
expression relop expression
```

where a relop is any of the six relational operators in C, and a matchop is either *~* (for contains) or *!~* (for does not contain). A conditional is an arithmetic expression, a relational expression, or a Boolean combination of these.

The special patterns BEGIN and END may be used to capture control before the first input line is read and after the last. BEGIN must be the first pattern, END the last.

A single character *c* may be used to separate the fields by starting the program with

```
BEGIN { FS = "c" }
```

or by using the *-Fc* option.

Other variable names with special meanings include NF, the number of fields in the current record; NR, the ordinal number of the current record; FILENAME, the name of the current input file; OFS, the output field separator (default blank); ORS, the output record separator (default newline); and OFMT, the output format for numbers (default "%.6g").

EXAMPLES

Print lines longer than 72 characters:

```
length > 72
```

Print first two fields in opposite order:

```
{ print $2, $1 }
```

Add up first column, print sum and average:

```
{ s += $1 }
END { print "sum is", s, " average is", s/NR }
```

Print fields in reverse order:

```
{ for (i = NF; i > 0; --i) print $i }
```


Print all lines between start/stop pairs:

```
/start/, /stop/
```

Print all lines whose first field is different from previous one:

```
$1 != prev { print; prev = $1 }
```

SEE ALSO

lex(1), sed(1)

A. V. Aho, B. W. Kernighan, P. J. Weinberger, *Awk - a pattern scanning and processing language*

BUGS

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number add 0 to it; to force it to be treated as a string concatenate "" to it.

REMARK

Awk exists on the CADMUS in 4 different versions: *awk.long*, *awk.fpp*, *awk.mot*, *awk.nsc*. The program *awk* is linked to *awk.fpp*. *Awk.long* is a version without floating point, which keeps all variables as long integers. The other 3 *awks* are compiled with the 3 different floating point versions (see *fpp*(3)). *Awk.nsc* will of course only work if your machine is equipped with an FPP board.

NAME

basename - strip filename affixes

SYNOPSIS

basename string [suffix]

DESCRIPTION

Basename deletes any prefix ending in '/' and the *suffix*, if present in *string*, from *string*, and prints the result on the standard output. It is normally used inside substitution marks `` in shell procedures.

This shell procedure invoked with the argument */usr/src/cmd/cat.c* compiles the named file and moves the output to *cat* in the current directory:

```
cc $1
```

```
mv a.out `basename $1 .c`
```

SEE ALSO

sh(1)

NAME

basic – Basic Interpreter

SYNOPSIS

basic [**-anrxtp**] [**+n**] [*file*] [*plotfile*] [*args* ...]

DESCRIPTION

Basic is an ANSI compatible Basic interpreter that runs under UNIX. It is written in C so it might not be quite as fast as other *Basic* interpreters.

The following options are available:

- a** abort (via IOT trap) on any error. Not for general use.
- n** requires that variable names follow the ANSI standard of a letter followed by an optional digit.
- p** (not yet supported) specifies the default plot output file or filter to be used for *hgr* and *hplot* commands.
- r** causes the *file* specified to be run. If **+n** is specified then execution will start at line *n*. *Basic* will exit upon completion of the program.
- t** is a debugging option that traces the operation of the interpreter.
- x** is a debugging option that suppresses the normal buffering of output.

file is an optional file name that specifies a program that is to be read (or run if **-r** was specified) before control is passed to the user.

args is an option list of arguments that are passed to the program via the *arg\$* function. The program may then use the argument strings as it sees fit.

BASIC COMMANDS

The following commands are available to control the running of a *Basic* program or interact with the system:

!command causes the UNIX command "command" to be executed, and control to be returned back to *Basic*.

AUTO [*line1*][*,line2*] causes the program to enter into automatic line numbering mode. Line numbering starts at *line1* and increments by *line2* for each line entered. Automatic numbering mode is terminated by a null line, an attention interrupt, or by any error. A star "*" will be printed if an automatically numbered line already exists in the program. To delete such a line enter a line with at least one space.

BYE causes *Basic* to exit.

CATALOG prints out the users current file catalog (equivalent to *!ls*).

CLEAR equivalent to *CLEAR VARS* plus *CLEAR STACK*. This is also done automatically as part of the *RUN* command.

CLEAR VARS clears the variable symbol table.

CLEAR STACK clears the execution stack. This is necessary after a "stack overflow" error, if *Basic* statements (such as "print i"), are to be used.

DEL [*line1*],[*line2*] deletes *line1* thru *line2* inclusive. It is also possible to delete a line by just specifying its line number.

DUMP displays the contents of the symbol table. This is mainly for debugging *Basic* itself.

EDIT causes the program in memory to be written out into a scratch file and the file editor (*\$EDITOR* shell variable) invoked upon it. Any changes made to the program will be reflected in the memory version upon leaving the editor.

HELP gives you some assistance. If you want more help use *man basic*.

LIST [*line1*],[*line2*] causes the program to be listed from lines *line1* thru *line2*. The default range is the entire program.

LOAD is the same as the *OLD* command.

MERGE [*file*] causes the program in *file* to be merged with the current program in memory. Existing lines will be replaced when duplicate line numbers are found.

MODE [*INT*][*FP*][*UNIX*][*IBM*] changes the prompting sequence.

OLD [*file*] causes the current program to be discarded, and a new program read in from *file*. Execution may then be begun via the *RUN* command. Note that all previous variable names are cleared.

RENUMBER [*line1*],[*line2*] causes the program in memory to be renumbered, starting at *line1*, with an increment of *line2*. *line1* defaults to 10, and *line2* to 10.

RUN [*line*] causes the program to start execution. If *line* is specified then execution starts at that line number.

SAVE [*file*] causes the program in memory to be saved in *file*. *file* defaults to the last name mentioned in a *OLD* or *SAVE* command.

SCR erases the program and then clears the symbol table.

TIMEOFF disables *TIMEON*.

TIMEON prints time spent in command.

UNSAVE [*file*] deletes (unlinks, removes) a file saved via the *SAVE* command.

BASIC STATEMENTS

The following *Basic* statements may be issued from a stored program or as a statement typed in immediate execution mode.

ASK is the same as the *INPUT* statement.

CHAIN file [,line] replace the current program with the program in the file *file*, and start execution at line *line* if present, otherwise at the first line of the program. Note that the *file* must either be a string expression or a string constant.

CLOSE #n closes unit *n*. It is quite legal to close a unit that has not been opened. When a program begins execution unit 0 is opened for input on standard input, and unit 1 is opened for output on standard output.

DATA constant-list consists of one or more constants separated by commas. Blanks or tabs are not modified in *DATA* statements.

DEF FN var(args) = expr causes a function (FN *var*) to be defined, with the given dummy arguments. When *FN var(...)* appears in an expression later the dummy arguments will have real values assigned to them and the value of the expression will be returned.

DIM var(bounds),... each of the variables specified is dimensioned with the bounds provided. *Basic*, as currently configured allows up to two dimensions for each vector variable. Note that a dimensioned variable is distinct from the scalar variable with the same name (e.g. "a" and "a(i)" are different, for all values of "i").

END causes the program to terminate. Some *Basic*'s require that *END* only appear on the last line of a program.

FOR var = expr1 TO expr2 [STEP expr3] the variable specified is assigned the value of *expr1*, the values of *expr2* and *expr3* (if specified) are evaluated and saved. The following statements up to the matching *next* are executed until the value of *var* is greater than *expr2*. When the *next* statement is executed the value *expr3* is added to *var*. *expr3* defaults to 1, and may be negative, in which case the *FOR* terminates when *var* is less than *expr2*.

GET is reserved for future expansion, and will allow input of binary files.

GOSUB line cause execution to transfer to the line number specified. When a *RETURN* is executed control will continue from the next statement after the *GOSUB*.

GOTO line causes execution to transfer to the line number specified.

IF expr THEN statement If the expression *expr* is *TRUE* (non-zero) then *statement* will be executed. Some *Basic*'s only allow a statement number or a *GOTO statement*, but here any legal *Basic* statement is allowed, including another *IF* statement. Other variations of the *IF* statement are

supported, where *THEN* is optional, and *statement* is a line number.

INPUT [*#n*,] [*string*,] *var*,... the variable (or variables) specified are read from the specified unit. If no unit is specified, unit 0 is used. If the string is specified it will be used as a prompt, as if printed via a *PRINT* statement.

LET var=expr assigns the value of *expr* to the variable *var*. Note that the *LET* is optional, but should be specified for compatibility with other *Basics*.

NEXT [*var*] terminates the preceeding *FOR* statement. The value of the for variable is incremented and if not greater than the limit the execution continues at the statement following the *FOR* statement.

NOTRACE disables the tracing started by the *TRACE* statement.

ON expr GOSUB line,... the value of *expr* is taken as an index into the list of line numbers and transfer is made to the *n*'th line number specified. Upon execution of a *RETURN* statement control will continue from the following statement.

ON expr GOTO line,... the value of *expr* is taken as an index into the list of line numbers and transfer is made to the *n*'th line number specified.

OPEN file FOR [INPUT/OUTPUT/APPEND] # n opens the file *file* for either input or output on unit *n*. *file* may be any string expression. The *input* statement reads (by default) from unit 0, the *print* statement writes by default to unit 1. *APPEND* is the same as *OUTPUT*, but writes at the end of the file.

PRINT [*#n*,] *expr*[,][:]... the value of the expression is printed on the specified unit. If no unit is specified then unit 1 is used. It may be either a number or a character string. If the comma is specified as a delimiter the values will be aligned in fields 16 columns wide. If a semi-colon is used there will be no spacing between fields. If there is no delimiter the last value will be followed by a new line. A *print* statement by itself will issue a *new line*.

PRINT [*#n*,] *USING fmt*; *expr*[,][:]... the value of the expression is printed in the same fashion as the *PRINT* statement, except that the arithmetic expression is written according to the format specification *fmt*. See the section *PRINT USING FORMAT SPECIFICATIONS*.

RANDOMIZE initializes the random number generator using the internal clock.

READ var,... will read values (from following *DATA* statements) into the variables given. An error will occur if there are insufficient values on *DATA* statements.

REM causes the following characters to be taken as a remark or

comment. Note that the same effect may be obtained by starting a line with either a single or double quote.

RESTORE causes the next *READ* statement to start at the first *DATA* statement in the program.

RETURN causes control to be transferred to the statement following the last *GOSUB* or *ON expr GOSUB statement*. If there is no enclosing *GOSUB* an error will occur.

STOP causes the program to terminate with an appropriate comment.

TRACE causes the line number of each statement to be printed as it is executed. This is sometimes helpful in following the execution of a program that is not behaving properly.

VARIABLES

In ANSI Basic, a variable must start with a letter, and may have an optional digit following. In UNIX *Basic* although only the first two characters of a variable name are considered significant, an arbitrarily long name may be used. The second character does not have to be a digit.

Following the name either a % (percent) or \$ (dollar) signs may be present. The percent sign indicates that the variable is to hold only integer values, while the dollar sign indicates that the variable is to hold string values. If neither is present then the variable will hold real (floating point) values.

FUNCTIONS

The following standard functions are provided: those marked with an asterisk (*) are not ANSI *Basic* and are a local implementation to facilitate communication with the UNIX operating system.

ABS(value) returns the absolute value of the given value.

ARG\$(n) * returns the n'th argument specified when *Basic* was opened. Argument zero is the name of the program file (if any). A null string will be returned if an argument is requested that was not specified. This is mostly useful when the *-r* switch is used to run a previously stored program.

ATN(value) returns the arc-tangent (in radians) function of the given value.

CHR\$(n) returns the ASCII character that corresponds to *n*. *n* should normally be an integer in the range $0 \leq n \leq 127$. Values in the range $128 \leq n \leq 255$ are possible. All other values are taken modulo 256.

COS(angle) returns the cosine function of the given angle in radians.

DATE\$() * returns the current date in the format *yy/mm/dd*.

EXP(value) returns the natural exponent of the given value.

INT(value) returns the integer value of the given value.

LEFT\$(string,length) the first *length* characters of the *string* are returned. It is an error if there are not *length* characters in *string*.

LOG(value) returns the natural log of the given value.

MID\$(string,start,length) the middle *length* characters of the *string* starting at offset *start* (1 = first character) are returned. It is an error if there are not *length* characters in *string* after *start*.

RND() returns a random number between 0.0 and 1.0.

RIGHT\$(string,length) the last *length* characters of the *string* are returned. It is an error if there are not *length* characters in *string*.

SIN(angle) returns the sine function of the given angle in radians.

SQR(value) returns the square root of the given value.

STR\$(value) returns a string corresponding to the value.

SYS(n) * executes system function "n", as follows:

- 1 turn off input echoing
- 2 turn on input echoing
- 3 get one character from keyboard
- 4 print symbol table
- 5 not implemented
- 6 not implemented
- 7 exit from *Basic* interpreter

SYSTEM(string) * executes *string* as a system (UNIX) command, and returns its exit status. Zero means the command terminated properly.

TAN(angle) returns the tangent function of the given angle in radians.

TIMES() returns the current time in hh:mm:ss format.

VAL(string) returns a value corresponding to the string.

OPERATORS

The following operators are available:

- + adds two numeric operands together. concatenates two strings.
- subtraction.

- * multiplication
- / division
- ^ exponentiation
- <> not equal comparison
- = equal comparison, also assignment operator
- < less than comparison
- > greater than comparison
- <= less than or equal comparison
- >= greater than or equal comparison
- AND logical AND of two comparisons
- OR logical OR of two comparisons

PRINT USING FORMAT SPECIFICATIONS

The *PRINT USING* statement format has the following general format (where [and] enclose optional features):

[+][**][\$\$]###[.]###[.]###.##[+]

The characters have the following meanings

- # indicates that a digit may be placed here.
- . indicates a decimal point
- + indicates a sign may be placed before (or after) the number.
- ** causes leading zero's to be replaced with asterisk fill, normally blanks are used.
- \$\$ causes a dollar sign to be placed at the start of the number.
- ,
- a comma is inserted into the number if any digits have been printed. Unlike some *Basic's* the position of the commas is significant.

FILES

- /bin/ed default editor
- /tmp/Ba* for the editor temporary file.

BUGS

The following features are not implemented:

- MAT operations.
- The "PRINT USING" implements only a minimal subset of the normal facilities.

NAME

bc - arbitrary-precision arithmetic language

SYNOPSIS

bc [-c] [-l] [file ...]

DESCRIPTION

Bc is an interactive processor for a language which resembles C but provides unlimited precision arithmetic. It takes input from any files given, then reads the standard input. The -l argument stands for the name of an arbitrary precision math library. The syntax for bc programs is as follows; L means letter a-z, E means expression, S means statement.

Comments

are enclosed in /* and */.

Names

simple variables: L

array elements: L [E]

The words 'ibase', 'obase', and 'scale'

Other operands

arbitrarily long numbers with optional sign and decimal point.

(E)

sqrt (E)

length (E) number of significant decimal digits

scale (E) number of digits right of decimal point

L (E , ... , E)

Operators

+ - * / % ^ (% is remainder; ^ is power)

++ -- (prefix and postfix; apply to names)

== <= >= != < >

= += -= *= /= %= ^=

Statements

E

{ S ; ... ; S }

if (E) S

while (E) S

for (E ; E ; E) S

null statement

break

quit

Function definitions

define L (L , ... , L) {

auto L , ... , L

S ; ... S

return (E)

}

Functions in -l math library

s(x) sine

c(x) cosine

e(x) exponential

l(x) log

a(x) arctangent
j(n,x) Bessel function

All function arguments are passed by value.

The value of a statement that is an expression is printed unless the main operator is an assignment. Either semicolons or newlines may separate statements. Assignment to *scale* influences the number of digits to be retained on arithmetic operations in the manner of *dc(1)*. Assignments to *ibase* or *obase* set the input and output number radix respectively.

The same letter may be used as an array, a function, and a simple variable simultaneously. All variables are global to the program. 'Auto' variables are pushed down during function calls. When using arrays as function arguments or defining them as automatic variables empty square brackets must follow the array name.

For example

```
scale = 20
define e(x){
    auto a, b, c, i, s
    a = 1
    b = 1
    s = 1
    for(i=1; 1==1; i++){
        a = a*x
        b = b*i
        c = a/b
        if(c == 0) return(s)
        s = s+c
    }
}
```

defines a function to compute an approximate value of the exponential function and

```
for(i=1; i<=10; i++) e(i)
```

prints approximate values of the exponential function of the first ten integers.

Bc is actually a preprocessor for *dc(1)*, which it invokes automatically, unless the *-c* (compile only) option is present. In this case the *dc* input is sent to the standard output instead.

FILES

/usr/lib/lib.b mathematical library
dc(1) desk calculator proper

SEE ALSO

dc(1)
L. L. Cherry and R. Morris, *BC - An arbitrary precision desk-calculator language*

BUGS

No *&&*, *||*, or *!* operators.
For statement must have all three E's.
Quit is interpreted when read, not when executed.

NAME

bdiff - big diff

SYNOPSIS

bdiff file1 file2 [*n*] [*-s*]

DESCRIPTION

Bdiff is used in a manner analogous to *diff*(1) to find which lines must be changed in two files to bring them into agreement. Its purpose is to allow processing of files which are too large for *diff*. *Bdiff* ignores lines common to the beginning of both files, splits the remainder of each file into *n*-line segments, and invokes *diff* upon corresponding segments. The value of *n* is 3500 by default. If the optional third argument is given, and it is numeric, it is used as the value for *n*. This is useful in those cases in which 3500-line segments are too large for *diff*, causing it to fail. If *file1* (*file2*) is -, the standard input is read. The optional *-s* (silent) argument specifies that no diagnostics are to be printed by *bdiff* (note, however, that this does not suppress possible exclamations by *diff*. If both optional arguments are specified, they must appear in the order indicated above.

The output of *bdiff* is exactly that of *diff*, with line numbers adjusted to account for the segmenting of the files (that is, to make it look as if the files had been processed whole). Note that because of the segmenting of the files, *bdiff* does not necessarily find a smallest sufficient set of file differences.

FILES

/tmp/bd????

SEE ALSO

diff(1).

DIAGNOSTICS

Use *help*(1) for explanations.

1. The first part of the report is a summary of the work done during the year. It is a brief statement of the facts and figures, and is intended to give a general impression of the work done. It is not a detailed account of the work, but a summary of the main results.

2. The second part of the report is a detailed account of the work done. It is a full and complete statement of the facts and figures, and is intended to give a detailed account of the work done. It is a full and complete statement of the facts and figures, and is intended to give a detailed account of the work done.

3. The third part of the report is a summary of the work done during the year. It is a brief statement of the facts and figures, and is intended to give a general impression of the work done. It is not a detailed account of the work, but a summary of the main results.

4. The fourth part of the report is a detailed account of the work done. It is a full and complete statement of the facts and figures, and is intended to give a detailed account of the work done. It is a full and complete statement of the facts and figures, and is intended to give a detailed account of the work done.

NAME

bfs - big file scanner

SYNOPSIS

bfs [-] name

DESCRIPTION

Bfs is (almost) like *ed*(1) except that it is read-only and processes much larger files. Files can be up to 1024K bytes (the maximum possible size) and 32K lines, with up to 255 characters per line. *Bfs* is usually more efficient than *ed* for scanning a file, since the file is not copied to a buffer. It is most useful for identifying sections of a large file where *csplit*(1) can be used to divide it into more manageable pieces for editing.

Normally, the size of the file being scanned is printed, as is the size of any file written with the *w* command. The optional - suppresses printing of sizes. Input is prompted with * if P and a carriage return are typed as in *ed*. Prompting can be turned off again by inputting another P and carriage return. Note that messages are given in response to errors if prompting is turned on.

All address expressions described under *ed* are supported. In addition, regular expressions may be surrounded with two symbols besides / and ?: > indicates downward search without wrap-around, and < indicates upward search without wrap-around. Since *bfs* uses a different regular expression-matching routine from *ed*, the regular expressions accepted are slightly wider in scope (see *regex*(3X)). There is a slight difference in mark names: only the letters a through z may be used, and all 26 marks are remembered.

The e, g, v, k, n, p, q, w, =, ! and null commands operate as described under *ed*. Commands such as —, +++-, +++=, -12, and +4p are accepted. Note that 1,10p and 1,10 will both print the first ten lines. The f command only prints the name of the file being scanned; there is no *remembered* file name. The w command is independent of output diversion, truncation, or crunching (see the xo, xt and xc commands, below). The following additional commands are available:

xf file

Further commands are taken from the named *file*. When an end-of-file is reached, an interrupt signal is received or an error occurs, reading resumes with the file containing the *xf*. *Xf* commands may be nested to a depth of 10.

xo [file]

Further output from the p and null commands is diverted to the named *file*, which, if necessary, is created mode 666. If *file* is missing, output is diverted to the standard output. Note that each diversion causes truncation or creation of the file.

: label

This positions a *label* in a command file. The *label* is terminated by new-line, and blanks between the : and the start of the *label* are ignored. This command may also be used to insert comments into a command file, since labels need not be referenced.

(...)xb/regular expression/label

A jump (either upward or downward) is made to *label* if the command succeeds. It fails under any of the following conditions:

1. Either address is not between 1 and 3.
2. The second address is less than the first.
3. The regular expression doesn't match at least one line in the specified range, including the first and last lines.

On success, . is set to the line matched and a jump is made to *label*. This command is the only one that doesn't issue an error message on bad addresses, so it may be used to test whether addresses are bad before other commands are executed. Note that the command

xb/^/ label

is an unconditional jump.

The **xb** command is allowed only if it is read from someplace other than a terminal. If it is read from a pipe only a downward jump is possible.

xt number

Output from the **p** and null commands is truncated to at most *number* characters. The initial number is 255.

xv[digit][spaces][value]

The variable name is the specified *digit* following the **xv**. **xv5100** or **xv5 100** both assign the value 100 to the variable 5. **Xv61,100p** assigns the value 1,100p to the variable 6. To reference a variable, put a **%** in front of the variable name. For example, using the above assignments for variables 5 and 6:

```
1,%5p
1,%5
%6
```

will all print the first 100 lines.

g/%5/p

would globally search for the characters 100 and print each line containing a match. To escape the special meaning of **%**, a **** must precede it.

g/".*\%[cds]/p

could be used to match and list lines containing *printf* of characters, decimal integers, or strings.

Another feature of the **xv** command is that the first line of output from a UNIX command can be stored into a variable. The only requirement is that the first character of *value* be an **!**. For example:

```
xv5!cat junk
!rm junk
!echo "%5"
```


xv6!expr %6 + 1

would put the current line into variable 5, print it, and increment the variable 6 by one. To escape the special meaning of ! as the first character of *value*, precede it with a \.

xv7\!date

stores the value !date into variable 7.

xbz label

xbn label

These two commands will test the last saved *return code* from the execution of a UNIX command (!*command*) or nonzero value, respectively, to the specified label. The two examples below both search for the next five lines containing the string *size*.

```
xv55
:1
/size/
xv5!expr %5 - 1
!if 0%5 != 0 exit 2
xbn 1
xv45
:1
/size/
xv4!expr %4 - 1
!if 0%4 = 0 exit 2
xbz 1
```

xc [switch]

If *switch* is 1, output from the *p* and null commands is crunched; if *switch* is 0 it isn't. Without an argument, *xc* reverses *switch*. Initially *switch* is set for no crunching. Crunched output has strings of tabs and blanks reduced to one blank and blank lines suppressed.

SEE ALSO

csplit(1), ed(1), regex(3X).

DIAGNOSTICS

? for errors in commands, if prompting is turned off. Self-explanatory error messages when prompting is on.

1000

1000

1000

1000

1000

1000

1000

1000

1000

1000

1000

1000

1000

NAME

bs — a compiler/interpreter for modest-sized programs

SYNOPSIS

bs [*file* [*args*]]

DESCRIPTION

Bs is a remote descendant of Basic and Snobol4 with a little C language thrown in. *Bs* is designed for programming tasks where program development time is as important as the resulting speed of execution. Formalities of data declaration and file/process manipulation are minimized. Line-at-a-time debugging, the *trace* and *dump* statements, and useful run-time error messages all simplify program testing. Furthermore, incomplete programs can be debugged; *inner* functions can be tested before *outer* functions have been written and vice versa.

If the command line *file* argument is provided, the file is used for input before the console is read. By default, statements read from the file argument are compiled for later execution. Likewise, statements entered from the console are normally executed immediately (see *compile* and *execute* below). Unless the final operation is assignment, the result of an immediate expression statement is printed.

Bs programs are made up of input lines. If the last character on a line is a \, the line is continued. *Bs* accepts lines of the following form:

statement
label statement

A label is a *name* (see below) followed by a colon. A label and a variable can have the same name.

A *bs* statement is either an expression or a keyword followed by zero or more expressions. Some keywords (*clear*, *compile*, *!*, *execute*, *include*, *ibase*, *obase*, and *run*) are always executed as they are compiled.

Statement Syntax:**expression**

The expression is executed for its side effects (value, assignment or function call). The details of expressions follow the description of statement types below.

break

Break exits from the inner-most *for/while* loop.

clear

Clears the symbol table and compiled statements. *Clear* is executed immediately.

compile [expression]

Succeeding statements are compiled (overrides the immediate execution default). The optional expression is evaluated and used as a file name for further input. A *clear* is associated with this latter case. *Compile* is executed immediately.

continue

Continue transfers to the loop-continuation of the current *for/while* loop.

dump

The name and current value of every non-local variable is printed. After an error or interrupt, the number of the last statement and (possibly) the user-function trace are displayed.

exit [expression]

Return to system level. The expression is returned as process status.

execute

Change to immediate execution mode (an interrupt has a similar effect). This statement does not cause stored statements to execute (see *run* below).

for name = expression expression statement**for** name = expression expression**next****for** expression , expression , expression statement**for** expression , expression , expression**next**

The *for* statement repetitively executes a statement (first form) or a group of statements (second form) under control of a named variable. The variable takes on the value of the first expression, then is incremented by one on each loop, not to exceed the value of the second expression. The third and fourth forms require three expressions separated by commas. The first of these is the initialization, the second is the test (true to continue), and the third is the loop-continuation action (normally an increment).

fun f([a, ...]) [v, ...]**nuf**

Fun defines the function name, arguments, and local variables for a user-written function. Up to ten arguments and local variables are allowed. Such names cannot be arrays, nor can they be I/O associated. Function definitions may not be nested.

freturn

A way to signal the failure of a user-written function. See the interrogation operator (?) below. If interrogation is not present, *freturn* merely returns zero. When interrogation is active, *freturn* transfers to that expression (possibly by-passing intermediate function returns).

ibase *N*

Ibase sets the input base (radix) to *N*. The only supported values for *N* are 8, 10 (the default), and 16. Hexadecimal values 10-15 are entered as a-f. A leading digit is required (i.e., f0a must be entered as 0f0a). *Ibase* (and *obase*, below) are executed immediately.

goto name

Control is passed to the internally stored statement with the matching label.

if expression statement

if expression

...
[**else**

...]

fi

The statement (first form) or group of statements (second form) is executed if the expression evaluates to non-zero. The strings 0 and "" (null) evaluate as zero. In the second form, an optional *else* allows for a group of statements to be executed when the first group is not. The only statement permitted on the same line with an *else* is an *if*; only other *fi*'s can be on the same line with a *fi*. The elision of *else* and *if* into an *elif* is supported. Only a single *fi* is required to close an *if* ... *elif* ... [*else* ...] sequence.

include expression

The expression must evaluate to a file name. The file must contain *bs* statements. Such statements become part of the program being compiled. *source* statements. *include* statements may not be nested.

obase *N*

Obase sets the input base to *N* (see *ibase* above).

onintr label

onintr

The *onintr* command provides program control of interrupts. In the first form, control will pass to the label given, just as if a *goto* had been executed at the time *onintr* was executed. The effect of the statement is cleared after each interrupt. In the second form, an interrupt will cause *bs* to terminate.

return [expression]

The expression is evaluated and the result is passed back as the value of a function call. If no expression is given, zero is returned.

run

The random number generator is reset. Control is passed to the first internal statement. If the *run* statement is contained in a file, it should be the last statement.

stop

Execution of internal statements is stopped. *Bs* reverts to immediate mode.

trace [expression]

The *trace* statement controls function tracing. If the expression is null (or evaluates to zero), tracing is turned off. Otherwise, a record of user-function calls/returns will be printed. Each *return* decrements the *trace* expression value.

while expression statement

while expression

...

next

While is similar to *for* except that only the conditional expression for loop-continuation is given.

! shell command

An immediate escape to the Shell.

...

This statement is ignored. It is used to interject commentary in a program.

Expression Syntax:

name

A name is used to specify a variable. Names are composed of a letter (upper or lower case) optionally followed by letters and digits. Only the first six characters of a name are significant. Except for names declared in *fun* statements, all names are global to the program. Names can take on numeric (double float) values, string values, or can be associated with input/output (see the built-in function *open()* below).

name ([expression [, expression] ...])

Functions can be called by a name followed by the arguments in parentheses separated by commas. Except for built-in functions (listed below), the name must be defined with a *fun* statement. Arguments to functions are passed by value.

name [expression [, expression] ...]

This syntax is used reference either arrays or tables (see built-in *table* functions below). For arrays, each expression is truncated to an integer and used as a specifier for the name. The resulting array reference is syntactically identical to a name; *a*[1,2] is the same as *a*[1][2]. The truncated expressions are restricted to values between 0 and 32767.

number

A number is used to represent a constant value. A number is written in Fortran style, and contains digits, an optional decimal point, and possibly a scale factor consisting of an *e* followed by a possibly signed exponent.

string

Character strings are delimited by " characters. The \ escape character allows the double quote (\"), new-line (\n), carriage return (\r), backspace (\b), and tab (\t) characters to appear in a string. Otherwise, \ stands for itself.

(expression)

Parentheses are used to alter the normal order of evaluation.

(expression, expression [, expression ...]) [expression]

The bracketed expression is used as a subscript to select a comma-separated expression from the parenthesized list. List elements are numbered from the left, starting at zero. The expression:

(False, True) [a == b]

has the value **True** if the comparison is true.

? expression

The interrogation operator tests for the success of the expression rather than its value. At the moment, it is useful for testing end-of-

file (see examples in the *Programming Tips* section below), the result of the *eval* built-in function, and for checking the return from user-written functions (see *freturn*). An interrogation "trap" (end-of-file, etc.) causes an immediate transfer to the most recent interrogation, possibly skipping assignment statements or intervening function levels.

- expression

The result is the negation of the expression.

++ name

Increments the value of the variable (or array reference). The result is the new value.

— name

Decrements the value of the variable. The result is the new value.

! expression

The logical negation of the expression. Watch out for the shell escape command.

expression *operator* expression

Common functions of two arguments are abbreviated by the two arguments separated by an operator denoting the function. Except for the assignment, concatenation, and relational operators, both operands are converted to numeric form before the function is applied.

Binary Operators (in increasing precedence):

=

= is the assignment operator. The left operand must be a name or an array element. The result is the right operand. Assignment binds right to left, all other operators bind left to right.

—

— (underscore) is the concatenation operator.

& |

& (logical and) has result zero if either of its arguments are zero. It has result one if both of its arguments are non-zero; | (logical or) has result zero if both of its arguments are zero. It has result one if either of its arguments is non-zero. Both operators treat a null string as a zero.

< <= > >= == !=

The relational operators (< less than, <= less than or equal, > greater than, >= greater than or equal, == equal to, != not equal to) return one if their arguments are in the specified relation. They return zero otherwise. Relational operators at the same level extend as follows: *a>b>c* is the same as *a>b & b>c*. A string comparison is made if both operands are strings.

+ -

Add and subtract.

* / %

Multiply, divide, and remainder.

^ Exponentiation.

Built-in Functions:

Dealing with arguments

arg(i)

is the value of the *i*-th actual parameter on the current level of function call. At level zero, *arg* returns the *i*-th command-line argument (*arg*(0) returns *bs*).

narg()

returns the number of arguments passed. At level zero, the command argument count is returned.

Mathematical

abs(x)

is the absolute value of *x*.

atan(x)

is the arctangent of *x*. Its value is between $-\pi/2$ and $\pi/2$.

ceil(x)

returns the smallest integer not less than *x*.

cos(x)

is the cosine of *x* (radians).

exp(x)

is the exponential function of *x*.

floor(x)

returns the largest integer not greater than *x*.

log(x)

is the natural logarithm of *x*.

rand()

is a uniformly distributed random number between zero and one.

sin(x)

is the sine of *x* (radians).

sqrt(x)

is the square root of *x*.

String operations

size(s)

the size (length in bytes) of *s* is returned.

format(f, a)

returns the formatted value of *a*. *F* is assumed to be a format specification in the style of *printf*(3S). Only the *%...f*, *%...e*, and *%...s* types are safe.

index(x, y)

returns the number of the first position in *x* that any of the characters from *y* matches. No match yields zero.

trans(s, f, t)

Translates characters of the source *s* from matching characters in *f*

to a character in the same position in *t*. Source characters that do not appear in *f* are copied to the result. If the string *f* is longer than *t*, source characters that match in the excess portion of *f* do not appear in the result.

substr(*s*, *start*, *width*)

returns the sub-string of *s* defined by the *starting* position and *width*.

match(*string*, *pattern*)

mstring(*n*)

The *pattern* is similar to the regular expression syntax of the *ed*(1) command. The characters *.*, *[*, *]*, *^* (inside brackets), *** and *\$* are special. The *mstring* function returns the *n*-th ($1 \leq n \leq 10$) substring of the subject that occurred between pairs of the pattern symbols *\(* and *\)* for the most recent call to *match*. To succeed, patterns must match the beginning of the string (as if all patterns began with *^*). The function returns the number of characters matched. For example:

```
match("a123ab123", ".*\([a-z]\)") == 6
mstring(1) == "b"
```

File handling

open(*name*, *file*, *function*)

close(*name*)

The *name* argument must be a *bs* variable name (passed as a string). For the *open*, the *file* argument may be 1) a 0 (zero), 1, or 2 representing standard input, output, or error output, respectively, 2) a string representing a file name, or 3) a string beginning with an *!* representing a command to be executed (via *sh -c*). The *function* argument must be either *r* (read), *w* (write), *W* (write without new-line), or *a* (append). After a *close*, the *name* reverts to being an ordinary variable. The initial associations are:

```
open("get", 0, "r")
open("put", 1, "w")
open("puterr", 2, "w")
```

Examples are given in the following section.

access(*s*, *m*)

executes *access*(2).

ftype(*s*)

returns a single character file type indication: *f* for regular file, *d* for directory, *b* for block special, or *c* for character special.

Tables

table(*name*, *size*)

A table in *bs* is an associatively accessed, single-dimension array. "Subscripts" (called keys) are strings (numbers are converted). The *name* argument must be a *bs* variable name (passed as a string). The *size* argument sets the minimum number of elements to be allocated. *Bs* prints an error message and stops on table overflow.

item(*name*, *i*)

key()

The *item* function accesses table elements sequentially (in normal use, there is no orderly progression of key values). Where the *item* function accesses values, the *key* function accesses the "subscript" of the previous *item* call. The *name* argument should not be quoted. Since exact table sizes are not defined, the interrogation operator should be used to detect end-of-table, for example:

```
table("t", 100)
```

```
...
# If word contains "party", the following expression adds one to
the count
# of that word:
++t[word]
...
# To print out the the key/value pairs:
for i = 0, ?(s = item(t, i)), ++i if key() put = key()_"":_s
```

iskey(name, word)

The *iskey* function tests whether the key *word* exists in the table *name* and returns one for true, zero for false.

*Odds and ends***eval(s)**

The string argument is evaluated as a *bs* expression. The function is handy for converting numeric strings to numeric internal form. *Eval* can also be used as a crude form of indirection, as in:

```
name = "xyz"
eval("++" _ name)
```

which increments the variable *xyz*. In addition, *eval* preceded by the interrogation operator permits the user to control *bs* error conditions. For example:

```
?eval("open(\"X\", \"XXX\", \"r\")")
```

returns the value zero if there is no file named "XXX" (instead of halting the user's program). The following executes a *goto* to the label *L* (if it exists):

```
label="L"
if !(?eval("goto " _ label)) puterr = "no label"
```

plot(request, args)

The *plot* function produces output on devices recognized by *tplot(1G)*. The *requests* are as follows:

Call

```
plot(0, term)
```

```
plot(1)
```

```
plot(2, string)
```

```
plot(3, x1, y1, x2, y2)
```

Function

causes further *plot* output to be piped into *tplot(1G)* with an argument of *-Tterm*.

"erases" the plotter.

labels the current point with *string*.

draws the line between (*x1,y1*) and (*x2,y2*).

plot(4, x, y, r)	draws a circle with center (x,y) and radius r.
plot(5, x1, y1, x2, y2, x3, y3)	draws an arc (counterclockwise) with center (x1,y1) and endpoints (x2,y2) and (x3,y3).
plot(6)	is not implemented.
plot(7, x, y)	makes the current point (x,y).
plot(8, x, y)	draws a line from the current point to (x,y).
plot(9, x, y)	draws a point at (x,y).
plot(10, string)	sets the line mode to string.
plot(11, x1, y1, x2, y2)	makes (x1,y1) the lower left corner of the plotting area and (x2,y2) the upper right corner of the plotting area.
plot(12, x1, y1, x2, y2)	causes subsequent x (y) coordinates to be multiplied by x1 (y1) and then added to x2 (y2) before they are plotted. The initial scaling is plot(12, 1.0, 1.0, 0.0, 0.0).

Some requests do not apply to all plotters. All requests except zero and twelve are implemented by piping characters to *tplot(1G)*. See *plot(5)* for more details.

last()

in immediate mode, *last* returns the most recently computed value.

PROGRAMMING TIPS

Using *bs* as a calculator:

```
$ bs
# Distance (inches) light travels in a nanosecond.
186000 * 5280 * 12 / 1e9
11.78496
...

# Compound interest (6% for 5 years on $1,000).
int = .06 / 4
bal = 1000
for i = 1 5*4 bal = bal + bal*int
bal - 1000
346.855007
...
exit
```

The outline of a typical *bs* program:

```
# initialize things:
var1 = 1
open("read", "infile", "r")
...
```



```
# compute:
while ?(str = read)
...
next
# clean up:
close("read")
...
# last statement executed (exit or stop):
exit
# last input line:
run
```

Input/Output examples:

```
# Copy "oldfile" to "newfile".
open("read", "oldfile", "r")
open("write", "newfile", "w")
...
while ?(write = read)
...
# close "read" and "write":
close("read")
close("write")

# Pipe between commands.
open("ls", "!ls *", "r")
open("pr", "!pr -2 -h 'List'", "w")
while ?(pr = ls) ...
...
# be sure to close (wait for) these:
close("ls")
close("pr")
```

SEE ALSO

ed(1), sh(1), tplot(1G), access(2), printf(3S), stdio(3S), Section 3 of this volume for further description of the mathematical functions (pow(3M) is used for exponentiation), plot(5). *Bs* uses the Standard Input/Output package.

REMARKS

Bs exists in 3 different versions on the CADMUS: *bs.fpp*, *bs.mot* and *bs.nsc*. The name *bs* is linked to *bs.fpp*. These three programs are compiled with the 3 different floating point versions of *cc*, see *fp*(3).

NAME

cal - print calendar

SYNOPSIS

cal [month] year

DESCRIPTION

Cal prints a calendar for the specified year. If a month is also specified, a calendar just for that month is printed. *Year* can be between 1 and 9999. The *month* is a number between 1 and 12. The calendar produced is that for England and her colonies.

Try September 1752.

BUGS

The year is always considered to start in January even though this is historically naive.

Beware that 'cal 78' refers to the early Christian era, not the 20th century.

Page 4

Page 5

Page 6

Page 7

Page 8

Page 9

Page 10

Page 11

Page 12

Page 13

NAME

calendar - reminder service

SYNOPSIS

calendar [-]

DESCRIPTION

Calendar consults the file **calendar** in the current directory and prints out lines that contain today's or tomorrow's date anywhere in the line. Most reasonable month-day dates such as "Dec. 7," "december 7," "12/7," etc., are recognized, but not "7 December" or "7/12". On weekends "tomorrow" extends through Monday.

When an argument is present, *calendar* does its job for every user who has a file **calendar** in his login directory and sends him any positive results by *mail*(1). Normally this is done daily in the wee hours under control of *cron*(8).

FILES

calendar
/usr/lib/calprog to figure out today's and tomorrow's dates
/etc/passwd
/tmp/cal*
/usr/lib/crontab

SEE ALSO

cron(8), mail(1).

BUGS

Your calendar must be public information for you to get reminder service.

Calendar's extended idea of "tomorrow" does not account for holidays.

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

NAME

`cat` - concatenate and print files

SYNOPSIS

`cat [-u] [-s] file ...`

DESCRIPTION

Cat reads each *file* in sequence and writes it on the standard output. Thus:

`cat file`

prints the file, and:

`cat file1 file2 >file3`

concatenates the first two files and places the result on the third.

If no input file is given, or if the argument `-` is encountered, *cat* reads from the standard input file. Output is buffered in 512-byte blocks unless the `-u` option is specified. The `-s` option makes *cat* silent about non-existent files. No input file may be the same as the output file unless it is a special file.

SEE ALSO

`cp(1)`, `pr(1)`.

NAME

cb - C program beautifier

SYNOPSIS

cb

DESCRIPTION

Cb places a copy of the C program from the standard input on the standard output with spacing and indentation that displays the structure of the program.

NAME

cc, lcc - C compiler

SYNOPSIS

cc [option] ... file ...

DESCRIPTION

Cc is the UNIX C compiler. It accepts several types of arguments:

Arguments whose names end with '.c' are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with '.o' substituted for '.c'. The '.o' file is normally deleted, however, if a single C program is compiled and linked all at one go. This C-compiler has the identifier "m68000" predefined, such that "#ifdef m68000" is true.

A call "cc test.c" is the same as "cc -c test.c && ld /lib/crt0.o test.o -lc && rm test.o". The call "cc *.o" is the same as "ld /lib/crt0.o *.o -lc". If your directory contains the file xyz.c, a call "make xyz" will result in "cc -o xyz xyz.c".

The following options are interpreted by cc. See ld(1) for link-time options.

-c Suppress the linking phase of the compilation, and force an object file to be produced even if only one program is compiled.

-o output

Name the final output file *output*. If this option is used the file 'a.out' will be left undisturbed. This option is passed on to the linker.

-f[FN] This option must be given, if the program includes floating point operations. The option -f alone will compile code for the Motorola Fast Floating Point Package, which is fast but does not have double precision arithmetic.

The option -fF will compile code for the Motorola IEEE Floating Point Package, and later for the Motorola floating point coprocessor. This package offers single and double precision according to the IEEE specifications. As long as the coprocessor is not available, the execution speed is rather slow.

The option -fN will compile code for the CADMUS FPP board with a NSC 16081 floating point processor. The numbers are also in IEEE format, but the generated code is much different from the one produced with option -fF.

With option -f, the library libffp.a will be searched before libc.a, with option -fF the library libmot.a, with option -fN the library libnsc.a. These libraries contain also the code which was formerly in the mathematical library libm.a. This library has been removed. Option -f makes the option -DFFP implicit. Option -fF makes the options -DIEEE and -DMOT_IEEE implicit. Option -fN makes the options -DIEEE and -DNSC_IEEE implicit.

-S Compile the named C programs, and produce an assembly listing on corresponding files suffixed '.s'. This option is used mainly for compiler debugging.

- L Arrange for the compiler to produce code which puts the current linenumber into the stackframe during execution. The C-stacktrace of *adb*(1) (command \$c or \$C) will then give for each active procedure the current linenumber.
- g Adds entries to the symbol table of the produced .o module which indicate line numbers and corresponding code location. To set a breakpoint at line 75 you can tell *adb* ":b L75". Make sure that only one module of a program is compiled with this option.
- p Arrange for the compiler to produce code which counts the number of times each routine is called; also, if linking takes place, replace the standard startup routine by one which automatically calls *monitor*(3C) at the start and arranges to write out a *mon.out* file at normal termination of execution of the object program. An execution profile can then be generated by use of *prof*(1).
- T With this option initialized data is put into the text- rather than the data segment. Thus for programs like the shell, whose text segments are shared between many users, memory space can be saved. Apply this option only to modules with nothing but constant data declarations.
- u All characters will be treated as "unsigned char".
- 4 Changes the integer size from 2 (default) to 4. This makes programs which neglect the differences between int and pointer or int and long much more portable. On the other hand, the produced code is less efficient. When the name *lcc* is linked to *cc*, *lcc* is equivalent to *cc -4*. This option is also passed to the linker. The linker transforms the option *-llib* to *-LLib*, e.g. *-lc* to *-LLc* or *-lcurses* to *-LLcurses*. These libraries are themselves compiled with option -4.
- 2 Like option -4. However, short parameters remain short and char parameters are only converted to short, not int, before they are pushed on the stack. This option is used for example to compile the system calls in the 4-byte-integer standard library */lib/libLc.a*, which must interface to a 2-byte-integer world.
- P Run only the macro preprocessor and place the result for each '.c' file in a corresponding '.i' file that has no '#' lines in it.
- E Run only the macro preprocessor and send the result to the standard output. The output is intended for compiler debugging; it is unacceptable as input to *cc*.
- Dname=def Define the *name* to the preprocessor, as if by '#define'. If no definition is given, the name is defined as 1.
- Dname Define the *name* to the preprocessor, as if by '#define'. If no definition is given, the name is defined as 1.
- Uname Remove any initial definition of *name*.
- Idir '#include' files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in

directories named in `-I` options, then in directories on a standard list.

`-Bstring`

Find substitute compiler passes in the files named *string* with the suffixes `cpp`, `c0`, `c1` and `c2`. If *string* is empty, use a standard backup version.

`-t[p012]`

Find only the designated compiler passes in the files whose names are constructed by a `-B` option. In the absence of a `-B` option, the *string* is taken to be `'/usr/src/cmd/c/'`.

Other arguments are taken to be either linker option arguments, or C-compatible object programs, typically produced by an earlier `cc` run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with name `a.out`.

FILES

<code>file.c</code>	input file
<code>file.o</code>	object file
<code>a.out</code>	linked output
<code>/tmp/ctm*</code>	temporaries for <code>cc</code>
<code>/lib/cpp</code>	preprocessor
<code>/lib/c[012]</code>	compiler passes for <code>cc</code>
<code>/lib/oc[012]</code>	backup compiler passes for <code>cc</code>
<code>/lib/ocpp</code>	backup preprocessor
<code>/lib/crt0.o</code>	runtime initialization
<code>/lib/lcrt0.o</code>	runtime initialization for option <code>-4</code>
<code>/lib/mlcrt0.o</code>	runtime initialization for profiling and option <code>-4</code>
<code>/lib/libc.a</code>	standard library, see <i>intro</i> (3)
<code>/lib/libLc.a</code>	standard library for option <code>-4</code>
<code>/lib/libffp.a</code>	library with fast floating point routines
<code>/lib/libmot.a</code>	library with IEEE floating point routines for Motorola package
<code>/lib/libnsc.a</code>	library with IEEE floating point routines for FPP board with NSC 16081
<code>/usr/include</code>	standard directory for <code>'#include'</code> files

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall, 1978
 D. M. Ritchie, *C Reference Manual*
`monitor(3C)`, `prof(1)`, `adb(1)`, `ld(1)`, `fp(3)`

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the linker.

... ..

... ..

... ..

... ..

... ..

... ..

... ..

... ..

... ..

... ..

... ..

... ..

... ..

... ..

... ..

... ..

... ..

NAME

cd - change working directory

SYNOPSIS

cd [directory]

DESCRIPTION

If *directory* is not specified, the value of shell parameter **\$HOME** is used as the new working directory. If *directory* specifies a complete path starting with /, .., *directory* becomes the new working directory. If neither case applies, *cd* tries to find the designated directory relative to one of the paths specified by the **\$CDPATH** shell variable. **\$CDPATH** has the same syntax as, and similar semantics to, the **\$PATH** shell variable. *Cd* must have execute (search) permission in *directory*.

Because a new process is created to execute each command, *cd* would be ineffective if it were written as a normal command; therefore, it is recognized and internal to the shell.

SEE ALSO

pwd(1), sh(1), chdir(2).

1. The first part of the document is a list of the names of the persons who have been named in the document.

2. The second part of the document is a list of the names of the persons who have been named in the document.

3. The third part of the document is a list of the names of the persons who have been named in the document.

4. The fourth part of the document is a list of the names of the persons who have been named in the document.

5. The fifth part of the document is a list of the names of the persons who have been named in the document. The names are listed in alphabetical order. The names are: [illegible]

6. The sixth part of the document is a list of the names of the persons who have been named in the document. The names are listed in alphabetical order. The names are: [illegible]

7. The seventh part of the document is a list of the names of the persons who have been named in the document.

8. The eighth part of the document is a list of the names of the persons who have been named in the document.

NAME

`cdc` - change the delta commentary of an SCCS delta

SYNOPSIS

`cdc -rSID [-m[mrlist]] [-y[comment]] files`

DESCRIPTION

`Cdc` changes the *delta commentary*, for the *SID* specified by the `-r` keyletter, of each named SCCS file.

Delta commentary is defined to be the Modification Request (**MR**) and comment information normally specified via the *delta*(1) command (`-m` and `-y` keyletters).

If a directory is named, `cdc` behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with *s*.) and unreadable files are silently ignored. If a name of `-` is given, the standard input is read (see *WARNINGS*); each line of the standard input is taken to be the name of an SCCS file to be processed.

Arguments to `cdc`, which may appear in any order, consist of *keyletter* arguments, and file names.

All the described *keyletter* arguments apply independently to each named file:

`-rSID` Used to specify the SCCS *ID*entification (*SID*) string of a delta for which the delta commentary is to be changed.

`-m[mrlist]` If the SCCS file has the *v* flag set (see *admin*(1)) then a list of **MR** numbers to be added and/or deleted in the delta commentary of the *SID* specified by the `-r` keyletter *may* be supplied. A null **MR** list has no effect.

MR entries are added to the list of **MRs** in the same manner as that of *delta*(1). In order to delete an **MR**, precede the **MR** number with the character *!* (see *EXAMPLES*). If the **MR** to be deleted is currently in the list of **MRs**, it is removed and changed into a "comment" line. A list of all deleted **MRs** is placed in the comment section of the delta commentary and preceded by a comment line stating that they were deleted.

If `-m` is not used and the standard input is a terminal, the prompt **MRs?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The **MRs?** prompt always precedes the **comments?** prompt (see `-y` keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the **MR** list.

Note that if the *v* flag has a value (see *admin*(1)), it is taken to be the name of a program (or shell procedure) which validates the correctness of the **MR**

numbers. If a non-zero exit status is returned from the **MR** number validation program, *cdc* terminates and the delta commentary remains unchanged.

-y[comment] Arbitrary text used to replace the *comment(s)* already existing for the delta specified by the **-r** keyletter. The previous comments are kept and preceded by a comment line stating that they were changed. A null *comment* has no effect.

If **-y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the *comment* text.

The exact permissions necessary to modify the SCCS file are documented in the *Source Code Control System User's Guide*. Simply stated, they are either (1) if you made the delta, you can change its delta commentary; or (2) if you own the file and directory you can modify the delta commentary.

EXAMPLES

```
cdc -r1.6 -m"bl78-12345 !bl77-54321 bl79-00001" -ytrouble s.file
```

adds bl78-12345 and bl79-00001 to the **MR** list, removes bl77-54321 from the **MR** list, and adds the comment **trouble** to delta 1.6 of s.file.

```
cdc -r1.6 s.file
```

```
MRs? !bl77-54321 bl78-12345 bl79-00001
```

```
comments? trouble
```

does the same thing.

WARNINGS

If SCCS file names are supplied to the *cdc* command via the standard input (**-** on the command line), then the **-m** and **-y** keyletters must also be used.

FILES

x-file (see *delta(1)*)

z-file (see *delta(1)*)

SEE ALSO

admin(1), *delta(1)*, *get(1)*, *help(1)*, *prs(1)*, *scsfile(5)*.

Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help(1)* for explanations.

NAME

checknews - check to see if user has news

SYNOPSIS

checknews [ynqevv] [readnews options]

DESCRIPTION

checknews reports to the user whether or not he has news.

y Reports "There is news" if the user has news to read.

n Reports "No news" if there isn't any news to read.

q causes checknews to be quiet. Instead of printing a message, the exit status indicates news. A status of 0 means no news, 1 means there is news.

v alters the y message to show the name of the first newsgroup containing unread news. Doubling v (e.g. vv) will cause an explanation of any claim of new news, and is useful if checknews and readnews disagree on whether there is news.

e Executes readnews(1) if there is news.

If there are no options, y is the default.

FILES

~/.newsrsrc Options and list of previously read articles

/usr/lib/news/active

Active newsgroups

SEE ALSO

inews(1), postnews(1), readnews(1)

NAME

`checknr` - check `nroff`/`troff` files

SYNOPSIS

```
checknr [ -s ] [ -f ] [ -a.x1.y1.x2.y2. ... .xn.yn ] [ -c.x1.x2.x3 ... .xn ]
[ file ... ]
```

DESCRIPTION

checknr checks a list of *nroff*(1) or *troff*(1) input files for certain kinds of errors involving mismatched opening and closing delimiters and unknown commands. If no files are specified, *checknr* checks the standard input. Delimiters checked are:

- (1) Font changes using `\fx ... \fP`.
- (2) Size changes using `\sx ... \s0`.
- (3) Macros that come in open ... close forms, for example, the `.TS` and `.TE` macros which must always come in pairs.

checknr knows about the *ms*(7) and *me*(7) macro packages.

Additional pairs of macros can be added to the list using the `-a` option. This must be followed by groups of six characters, each group defining a pair of macros. The six characters are a period, the first macro name, another period, and the second macro name. For example, to define a pair `.BS` and `.ES`, use `-a.BS.ES`

The `-c` option defines commands which would otherwise be complained about as undefined.

The `-f` option requests *checknr* to ignore `\f` font changes.

The `-s` option requests *checknr* to ignore `\s` size changes.

checknr is intended to be used on documents that are prepared with *checknr* in mind, much the same as *lint*. It expects a certain document writing style for `\f` and `\s` commands, in that each `\fx` must be terminated with `\fP` and each `\sx` must be terminated with `\s0`. While it will work to directly go into the next font or explicitly specify the original font or point size, and many existing documents actually do this, such a practice will produce complaints from *checknr*. Since it is probably better to use the `\fP` and `\s0` forms anyway, you should think of this as a contribution to your document preparation style.

SEE ALSO

`nroff`(1), `troff`(1), *ms*(7), *me*(7), `checkeqn`(1)

DIAGNOSTICS

Complaints about unmatched delimiters.

Complaints about unrecognized commands.

Various complaints about the syntax of commands.

AUTHOR

Mark Horton

BUGS

There is no way to define a 1 character macro name using `-a`

SECRET

SECRET - CONFIDENTIAL

SECRET

SECRET - CONFIDENTIAL

SECRET

SECRET

SECRET - CONFIDENTIAL

SECRET

SECRET

SECRET - CONFIDENTIAL

SECRET

SECRET - CONFIDENTIAL

SECRET

SECRET

SECRET

SECRET - CONFIDENTIAL

SECRET

SECRET

SECRET

SECRET

SECRET

SECRET

SECRET

SECRET

SECRET - CONFIDENTIAL

SECRET

SECRET

NAME

chfn - change full name of user

SYNOPSIS

chfn name string

DESCRIPTION

Chfn is a command similar to *passwd*(1) except that it is used to change the *gecos* field of the password file rather than the password entry. Note that the string specified will replace the entire *gecos* field of the specified user. If (as on the UCB system) this field contains information in addition to the user's full name, this information must be included in string or it will be deleted. Hence *chfn* can be used to fix phone numbers, offices, etc.

An example use of this command would be

chfn mark '& Horton,508E,7686,5240633'

Note that the string must in general be quoted to shield blanks and special characters from the shell. The field should consist of the users name, followed by their office number, followed by the last 4 digits of their office extension and finally their home phone number. Any of these can be omitted. There should be no spaces in the entry except for those in your name.

It is a good idea to run *finger*(1) on the user before and after *chfn* to make sure you have formatted the data correctly.

SEE ALSO

finger(1), *passwd*(5), *passwd*(1)

AUTHOR

Mark Horton

BUGS

The encoding of the office and extension information is installation dependent.

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

NAME

chmod - change mode

SYNOPSIS

chmod mode file ...

DESCRIPTION

The permissions of each named file are changed according to *mode*, which may be absolute or symbolic. An absolute *mode* is an octal number constructed from the OR of the following modes:

4000	set user ID on execution
2000	set group ID on execution
1000	sticky bit, see <i>chmod(2)</i>
0400	read by owner
0200	write by owner
0100	execute (search in directory) by owner
0070	read, write, execute (search) by group
0007	read, write, execute (search) by others

A symbolic *mode* has the form:

[*who*] *op permission* [*op permission*]

The *who* part is a combination of the letters *u* (for user's permissions), *g* (group) and *o* (other). The letter *a* stands for *ugo*, the default if *who* is omitted.

Op can be *+* to add *permission* to the file's mode, *-* to take away *permission*, or *=* to assign *permission* absolutely (all other bits will be reset).

Permission is any combination of the letters *r* (read), *w* (write), *x* (execute), *s* (set owner or group ID) and *t* (save text - sticky); *u*, *g* or *o* indicate that *permission* is to be taken from the current mode. Omitting *permission* is only useful with *=* to take away all permissions.

Multiple symbolic modes separated by commas may be given. Operations are performed in the order specified. The letter *s* is only useful with *u* or *g* and *t* only works with *u*.

Only the owner of a file (or the super-user) may change its mode.

EXAMPLES

The first example denies write permission to others, the second makes a file executable, the third makes a file readable and executable by everybody, and writable by the owner.

chmod o-w file

chmod +x file

chmod 0755 file

SEE ALSO

ls(1), chmod(2).

NAME

chown, chgrp - change owner or group

SYNOPSIS

chown owner file ...

chgrp group file ...

DESCRIPTION

Chown changes the owner of the *files* to *owner*. The owner may be either a decimal user ID or a login name found in the password file.

Chgrp changes the group ID of the *files* to *group*. The group may be either a decimal group ID or a group name found in the group file.

FILES

/etc/passwd

/etc/group

SEE ALSO

chown(2), group(5), passwd(5).

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

CLEAR(1)

MUNIX

CLEAR(1)

NAME

clear - clear terminal screen

SYNOPSIS

clear

DESCRIPTION

Clear clears your screen if this is possible. It looks in the environment for the terminal type and then in */etc/termcap* to figure out how to clear the screen.

FILES

/etc/termcap terminal capability data base

10/23/77

10/23/77

10/23/77

10/23/77

10/23/77

10/23/77

10/23/77

10/23/77

10/23/77

10/23/77

NAME

`cmp` - compare two files

SYNOPSIS

`cmp [-l] [-s] file1 file2`

DESCRIPTION

The two files are compared. (If *file1* is `-`, the standard input is used.) Under default options, *cmp* makes no comment if the files are the same; if they differ, it announces the byte and line number at which the difference occurred. If one file is an initial subsequence of the other, that fact is noted.

Options:

- `-l` Print the byte number (decimal) and the differing bytes (octal) for each difference.
- `-s` Print nothing for differing files; return codes only.

SEE ALSO

`comm(1)`, `diff(1)`.

DIAGNOSTICS

Exit code 0 is returned for identical files, 1 for different files, and 2 for an inaccessible or missing argument.

1940

1940

1940

1940

1940

1940

1940

1940

1940

1940

1940

1940

1940

1940

1940

NAME

`col` - filter reverse line feeds

SYNOPSIS

`col [-bfx]`

DESCRIPTION

`Col` reads the standard input and writes the standard output. It performs the line overlays implied by reverse line feeds (ESC-7 in ASCII) and by forward and reverse half line feeds (ESC-9 and ESC-8). `Col` is particularly useful for filtering multicolumn output made with the `.rt` command of `troff` and output resulting from use of the `tbl(1)` preprocessor.

Although `col` accepts half line motions in its input, it normally does not emit them on output. Instead, text that would appear between lines is moved to the next lower full line boundary. This treatment can be suppressed by the `-f` (fine) option; in this case the output from `col` may contain forward half line feeds (ESC-9), but will still never contain either kind of reverse line motion.

If the `-b` option is given, `col` assumes that the output device in use is not capable of backspacing. In this case, if several characters are to appear in the same place, only the last one read will be taken.

The control characters SO (ASCII code 017), and SI (016) are assumed to start and end text in an alternate character set. The character set (primary or alternate) associated with each printing character read is remembered; on output, SO and SI characters are generated where necessary to maintain the correct treatment of each character.

`Col` normally converts white space to tabs to shorten printing time. If the `-x` option is given, this conversion is suppressed.

All control characters are removed from the input except space, backspace, tab, return, newline, ESC (033) followed by one of 789, SI, SO, and VT (013). This last character is an alternate form of full reverse line feed, for compatibility with some other hardware conventions. All other non-printing characters are ignored.

SEE ALSO

`troff(1)`, `tbl(1)`, `greek(1)`

BUGS

Can't back up more than 128 lines.

No more than 800 characters, including backspaces, on a line.

11/10/68

11/10/68

11/10/68

11/10/68

11/10/68

11/10/68

11/10/68

11/10/68

11/10/68

11/10/68

11/10/68

11/10/68

11/10/68

11/10/68

11/10/68

11/10/68

NAME

`colcrt` - filter `nroff` output for CRT previewing

SYNOPSIS

`colcrt` [-] [-2] [file ...]

DESCRIPTION

Colcrt provides virtual half-line and reverse line feed sequences for terminals without such capability, and on which overstriking is destructive. Half-line characters and underlining (changed to dashing '-') are placed on new lines in between the normal output lines.

The optional - suppresses all underlining. It is especially useful for previewing *allboxed* tables from *tbl*(1).

The option -2 causes all half-lines to be printed, effectively double spacing the output. Normally, a minimal space output format is used which will suppress empty lines. The program never suppresses two consecutive empty lines, however. The -2 option is useful for sending output to the line printer when the output contains superscripts and subscripts which would otherwise be invisible.

A typical use of *colcrt* would be

```
'tbl exum2.n | nroff -ms | colcrt - | more
```

SEE ALSO

`nroff/troff`(1), `col`(1), `more`(1), `ul`(1)

AUTHOR

William Joy

BUGS

Should fold underlines onto blanks even with the '-' option so that a true underline character would show; if we did this, however, *colcrt* wouldn't get rid of *cu'd* underlining completely.

Can't back up more than 102 lines.

General overstriking is lost; as a special case '|' overstruck with '-' or underline becomes '+'.
Lines are trimmed to 132 characters.

Some provision should be made for processing superscripts and subscripts in documents which are already double-spaced.

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

131-1000

NAME

`colrm` - remove columns from a file

SYNOPSIS

`colrm [startcol [endcol]]`

DESCRIPTION

Colrm removes selected columns from a file. Input is taken from standard input. Output is sent to standard output.

If called with one parameter the columns of each line will be removed starting with the specified column. If called with two parameters the columns from the first column to the last column will be removed.

Column numbering starts with column 1.

SEE ALSO

`expand(1)`

AUTHOR

Jeff Schriebman

10000

10000

10000

10000

10000

10000

10000

10000

10000

10000

10000

10000

10000

NAME

comb - combine SCCS deltas

SYNOPSIS

comb [-o] [-s] [-psid] [-clist] files

DESCRIPTION

Comb generates a shell procedure (see *sh*(1)) which, when run, will reconstruct the given SCCS files. The reconstructed files will, hopefully, be smaller than the original files. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *comb* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with *s*.) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The generated shell procedure is written on the standard output.

The keyletter arguments are as follows. Each is explained as though only one named file is to be processed, but the effects of any keyletter argument apply independently to each named file.

-psid The SCCS IDentification string (SID) of the oldest delta to be preserved. All older deltas are discarded in the reconstructed file.

-clist A list (see *get*(1) for the syntax of a list) of deltas to be preserved. All other deltas are discarded.

-o For each *get -e* generated, this argument causes the reconstructed file to be accessed at the release of the delta to be created, otherwise the reconstructed file would be accessed at the most recent ancestor. Use of the -o keyletter may decrease the size of the reconstructed SCCS file. It may also alter the shape of the delta tree of the original file.

-s This argument causes *comb* to generate a shell procedure which, when run, will produce a report giving, for each file: the file name, size (in blocks) after combining, original size (also in blocks), and percentage change computed by:

$$100 * (\text{original} - \text{combined}) / \text{original}$$

It is recommended that before any SCCS files are actually combined, one should use this option to determine exactly how much space is saved by the combining process.

If no keyletter arguments are specified, *comb* will preserve only leaf deltas and the minimal number of ancestors needed to preserve the tree.

FILES

s.COMB The name of the reconstructed SCCS file.
comb????? Temporary.

SEE ALSO

admin(1), delta(1), get(1), help(1), prs(1), sccsfile(5).

Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Comb may rearrange the shape of the tree of deltas. It may not save any space; in fact, it is possible for the reconstructed file to actually be larger than the original.

NAME

`comm` - select or reject lines common to two sorted files

SYNOPSIS

`comm` [- [123]] *file1 file2*

DESCRIPTION

Comm reads *file1* and *file2*, which should be ordered in ASCII collating sequence, and produces a three column output: lines only in *file1*; lines only in *file2*; and lines in both files. The filename '-' means the standard input.

Flags 1, 2, or 3 suppress printing of the corresponding column. Thus `comm -12` prints only the lines common to the two files; `comm -23` prints only lines in the first file but not in the second; `comm -123` is a no-op.

SEE ALSO

`cmp(1)`, `diff(1)`, `uniq(1)`

1973

1974

1975

1976

1977

1978

1979

1980

1981

1982

1983

1984

1985

1986

1987

1988

1989

1990

1991

1992

1993

1994

1995

1996

1997

1998

1999

2000

2001

2002

2003

2004

2005

2006

2007

2008

2009

2010

2011

2012

2013

2014

2015

2016

2017

2018

NAME

`compact`, `uncompact`, `ccat` - compress and uncompress files, and cat them

SYNOPSIS

```
compact [ name ... ]  
uncompact [ name ... ]  
ccat [ file ... ]
```

DESCRIPTION

Compact compresses the named files using an adaptive Huffman code. If no file names are given, then the standard input is compacted to the standard output. *Compact* operates as an on-line algorithm. Each time a byte is read, it is encoded immediately according to the current prefix code. This code is an optimal Huffman code for the set of frequencies seen so far. It is unnecessary to prepend a decoding tree to the compressed file since the encoder and the decoder start in the same state and stay synchronized. Furthermore, *compact* and *uncompact* can operate as filters. In particular,

... | `compact` | `uncompact` | ...

operates as a (very slow) no-op.

When an argument *file* is given, it is compacted and the resulting file is placed in *file.C*; *file* is unlinked. The first two bytes of the compacted file code the fact that the file is compacted. This code is used to prohibit recompaction.

The amount of compression to be expected depends on the type of file being compressed. Typical values of compression are: Text (38%), Pascal Source (43%), C Source (36%) and Binary (19%). These values are the percentages of file bytes reduced.

Uncompact restores the original file from a file compressed by *compact*. If no file names are given, then the standard input is uncompact to the standard output.

Ccat cats the original file from a file compressed by *compact*, without uncompressing the file.

RESTRICTION

The last segment of the filename must contain fewer than thirteen characters to allow space for the appended '.C'.

FILES

*.C compacted file created by *compact*, removed by *uncompact*

SEE ALSO

Gallager, Robert G., "Variations on a Theme of Huffman", *I.E.E.E. Transactions on Information Theory*, vol. IT-24, no. 6, November 1978, pp. 668 - 674.

AUTHOR

Colin L. Mc Master

The first part of the report deals with the general situation of the country. It is a very interesting and informative study of the country's development. The second part of the report deals with the specific details of the country's development. It is a very detailed and thorough study of the country's development. The third part of the report deals with the specific details of the country's development. It is a very detailed and thorough study of the country's development.

The fourth part of the report deals with the specific details of the country's development. It is a very detailed and thorough study of the country's development. The fifth part of the report deals with the specific details of the country's development. It is a very detailed and thorough study of the country's development. The sixth part of the report deals with the specific details of the country's development. It is a very detailed and thorough study of the country's development.

The seventh part of the report deals with the specific details of the country's development. It is a very detailed and thorough study of the country's development. The eighth part of the report deals with the specific details of the country's development. It is a very detailed and thorough study of the country's development. The ninth part of the report deals with the specific details of the country's development. It is a very detailed and thorough study of the country's development.

NAME

`cp`, `ln`, `mv` - copy, link or move files

SYNOPSIS

```
cp file1 [ file2 ...] target  
ln file1 [ file2 ...] target  
mv file1 [ file2 ...] target
```

DESCRIPTION

File1 is copied (linked, moved) to *target*. Under no circumstance can *file1* and *target* be the same. If *target* is a directory, then one or more files are copied (linked, moved) to that directory.

If *mv* determines that the mode of *target* forbids writing, it will print the mode (see *chmod*(2)) and read the standard input for one line (if the standard input is a terminal); if the line begins with *y*, the move takes place; if not, *mv* exits.

Only *mv* will allow *file1* to be a directory, in which case the directory rename will occur only if the two directories have the same parent.

SEE ALSO

cpio(1), *link*(8), *rm*(1), *chmod*(2).

BUGS

If *file1* and *target* lie on different file systems, *mv* must copy the file and delete the original. In this case the owner name becomes that of the copying process and any linking relationship with other files is lost.

Ln will not link across file systems.

1000

1000

1000

1000

1000

1000

NAME

`cpio` - copy file archives in and out

SYNOPSIS

`cpio -o [acBSv]`

`cpio -i [BScdmrtuvfshb6] [patterns]`

`cpio -p [adlmruv] directory`

DESCRIPTION

`Cpio -o` (copy out) reads the standard input to obtain a list of path names and copies those files onto the standard output together with path name and status information.

`Cpio -i` (copy in) extracts files from the standard input which is assumed to be the product of a previous `cpio -o`. Only files with names that match *patterns* are selected. *Patterns* are given in the name-generating notation of `sh(1)`. In *patterns*, meta-characters `?`, `*`, and `[...]` match the slash `/` character. Multiple *patterns* may be specified and if no *patterns* are specified, the default for *patterns* is `*` (i.e., select all files). The extracted files are conditionally created and copied into the current directory tree based upon the options described below.

`Cpio -p` (pass) reads the standard input to obtain a list of path names of files that are conditionally created and copied into the destination *directory* tree based upon the options described below.

The meanings of the available options are:

- a** Reset access times of input files after they have been copied.
- B** Input/output is to be blocked 5,120 bytes to the record (does not apply to the *pass* option; meaningful only with data directed to or from `/dev/rmt?`).
- S** Input/output is to be blocked 61,440 bytes to the record (does not apply to the *pass* option; meaningful for streaming tapes, e.g. `/dev/rst0`).
- d** *Directories* are to be created as needed.
- c** Write *header* information in ASCII character form for portability.
- r** Interactively *rename* files. If the user types a null line, the file is skipped.
- t** Print a *table of contents* of the input. No files are created.
- u** Copy *unconditionally* (normally, an older file will not replace a newer file with the same name).
- v** *Verbose*: causes a list of file names to be printed. When used with the **t** option, the table of contents looks like the output of an `ls -l` command (see `ls(1)`).
- l** Whenever possible, link files rather than copying them. Usable only with the `-p` option.
- m** Retain previous file modification time. This option is ineffective on directories that are being copied.
- f** Copy in all files except those in *patterns*.
- s** Swap bytes. Use only with the `-i` option.
- h** Swap halfwords. Use only with the `-i` option.
- b** Swap both bytes and halfwords. Use only with the `-i` option.
- 6** Process an old (i.e., UNIX System *Sixth* Edition format) file. Only useful with `-i` (copy in).

EXAMPLES

The first example below copies the contents of a directory into an archive; the second duplicates a directory hierarchy:

```
ls | cpio -o >/dev/rmt0
```

```
cd olddir
```

```
find . -depth -print | cpio -pdl newdir
```

The trivial case "find . -depth -print | cpio -oB >/dev/rmt0" can be handled more efficiently by:

```
find . -cpio /dev/rmt0
```

SEE ALSO

ar(1), find(1), cpio(5).

BUGS

Path names are restricted to 128 characters. If there are too many unique linked files, the program runs out of memory to keep track of them and, thereafter, linking information is lost. Only the super-user can copy special files.

NAME

`cpp` - the C language preprocessor

SYNOPSIS

`/lib/cpp [option ...] [ifile [ofile]]`

DESCRIPTION

`Cpp` is the C language preprocessor which is invoked as the first pass of any C compilation using the `cc(1)` command. Thus the output of `cpp` is designed to be in a form acceptable as input to the next pass of the C compiler. As the C language evolves, `cpp` and the rest of the C compilation package will be modified to follow these changes. Therefore, the use of `cpp` other than in this framework is not suggested. The preferred way to invoke `cpp` is through the `cc(1)` command since the functionality of `cpp` may someday be moved elsewhere. See `m4(1)` for a general macro processor.

`Cpp` optionally accepts two file names as arguments. *ifile* and *ofile* are respectively the input and output for the preprocessor. They default to standard input and standard output if not supplied.

The following *options* to `cpp` are recognized:

- P Preprocess the input without producing the line control information used by the next pass of the C compiler.
- C By default, `cpp` strips C-style comments. If the `-C` option is specified, all comments (except those found on `cpp` directive lines) are passed along.

-U*name*

Remove any initial definition of *name*, where *name* is a reserved symbol that is predefined by the particular preprocessor. The current list of these possibly reserved symbols includes:

operating system: `ibm, gcos, os, tss, unix`
 hardware: `interdata, pdp11, u370, u3b, vax`
 UNIX System variant:
 `RES, RT`

-D*name*-D*name*=*def*

Define *name* as if by a `#define` directive. If no `=def` is given, *name* is defined as 1.

- Idir Change the algorithm for searching for `#include` files whose names do not begin with `/` to look in *dir* before looking in the directories on the standard list. Thus, `#include` files whose names are enclosed in `" "` will be searched for first in the directory of the *ifile* argument, then in directories named in `-I` options, and last in directories on a standard list. For `#include` files whose names are enclosed in `<>`, the directory of the *ifile* argument is not searched.

Two special names are understood by `cpp`. The name `__LINE__` is defined as the current line number (as a decimal integer) as known by `cpp`, and `__FILE__` is defined as the current file name (as a C string) as known by `cpp`. They can be used anywhere (including in macros) just as any other defined name.

All *cpp* directives start with lines begun by *#*. The directives are:

#define name token-string

Replace subsequent instances of *name* with *token-string*.

#define name(arg, ..., arg) token-string

Notice that there can be no space between *name* and the (. Replace subsequent instances of *name* followed by a (, a list of comma separated tokens, and a) by *token-string* where each occurrence of an *arg* in the *token-string* is replaced by the corresponding token in the comma separated list.

#undef name

Cause the definition of *name* (if any) to be forgotten from now on.

#include "filename"

#include <filename>

Include at this point the contents of *filename* (which will then be run through *cpp*). When the *<filename>* notation is used, *filename* is only searched for in the standard places. See the *-I* option above for more detail.

#line integer-constant "filename"

Causes *cpp* to generate line control information for the next pass of the C compiler. *Integer-constant* is the line number of the next line and *filename* is the file where it comes from. If *"filename"* is not given, the current file name is unchanged.

#endif

Ends a section of lines begun by a test directive (*#if*, *#ifdef*, or *#ifndef*). Each test directive must have a matching *#endif*.

#ifdef name

The lines following will appear in the output if and only if *name* has been the subject of a previous *#define* without being the subject of an intervening *#undef*.

#ifndef name

The lines following will not appear in the output if and only if *name* has been the subject of a previous *#define* without being the subject of an intervening *#undef*.

#if constant-expression

Lines following will appear in the output if and only if the *constant-expression* evaluates to non-zero. All binary non-assignment C operators, the *?:* operator, the unary *-*, *!*, and *~* operators are all legal in *constant-expression*. The precedence of the operators is the same as defined by the C language. There is also a unary operator *defined*, which can be used in *constant-expression* in these two forms: *defined (name)* or *defined name*. This allows the utility of *#ifdef* and *#ifndef* in a *#if* directive. Only these operators, integer constants, and names which are known by *cpp* should be used in *constant-expression*. In particular, the *sizeof* operator is not available.

#else Reverses the notion of the test directive which matches this directive. So if lines previous to this directive are ignored, the following lines will appear in the output. And vice versa.

The test directives and the possible **#else** directives can be nested.

FILES

/usr/include standard directory for **#include** files

SEE ALSO

cc(1), m4(1).

DIAGNOSTICS

The error messages produced by *cpp* are intended to be self-explanatory. The line number and filename where the error occurred are printed along with the diagnostic.

NOTES

When newline characters were found in argument lists for macros to be expanded, previous versions of *cpp* put out the newlines as they were found and expanded. The current version of *cpp* replaces these newlines with blanks to alleviate problems that the previous versions had when this occurred.

1. The first part of the report is a general introduction to the subject.

2. The second part of the report is a detailed description of the methods used in the study.

3. The third part of the report is a discussion of the results of the study.

4. The fourth part of the report is a conclusion and a list of references.

NAME

create - create a file

SYNOPSIS

create *file...*

DESCRIPTION

Create creates one or more files named *file...* with a length of 0 bytes. This is written as a shell procedure and should really give you a laugh when you read it.

11/11/11

11/11/11

11/11/11

11/11/11

11/11/11

11/11/11

11/11/11

11/11/11

11/11/11

11/11/11

11/11/11

NAME

`cref` - make cross-reference listing

SYNOPSIS

`cref [-acilnostux123] files`

DESCRIPTION

Cref makes a cross-reference listing of assembler or C programs; *files* are searched for symbols in the appropriate syntax.

The output report is in four columns:

1. symbol;
2. file name;
3. see below;
4. text as it appears in the file.

Cref uses either an *ignore* file or an *only* file. If the `-i` option is given, the next argument is taken to be an *ignore* file; if the `-o` option is given, the next argument is taken to be an *only* file. *Ignore* and *only* files are lists of symbols separated by new-lines. All symbols in an *ignore* file are ignored in columns 1 and 3 of the output. If an *only* file is given, only symbols in that file will appear in column 1. Only one of these options may be given; the default setting is `-i` using the default ignore file (see *FILES* below). Assembler pre-defined symbols or C keywords are ignored.

The `-s` option causes current symbols to be put in column 3. In the assembler, the current symbol is the most recent name symbol; in C, the current function name. The `-l` option causes the line number within the file to be put in column 3.

The `-t` option causes the next available argument to be used as the name of the intermediate file (instead of the temporary file `/tmp/crt??`). This file is created and is *not* removed at the end of the process.

The *cref* options are:

- `a` assembler format (default)
- `c` C format input
- `i` use an *ignore* file (see above)
- `l` put line number in column 3 (instead of current symbol)
- `n` omit column 4 (no context)
- `o` use an *only* file (see above)
- `s` current symbol in column 3 (default)
- `t` user-supplied temporary file
- `u` print only symbols that occur exactly once
- `x` print only C external symbols
- `1` sort output on column 1 (default)
- `2` sort output on column 2
- `3` sort output on column 3.

FILES

`/tmp/crt??` temporaries
`/usr/lib/cref/aign` default assembler *ignore* file
`/usr/lib/cref/atab` grammar table for assembler files

/usr/lib/cref/cign
default C ignore file
/usr/lib/cref/ctab
grammar table for C files
/usr/lib/cref/crpost
post-processor
/usr/lib/cref/upost
post-processor for **-u** option

SEE ALSO

as(1), cc(1), sort(1), xref(1).

BUGS

Cref inserts an ASCII DEL character into the intermediate file after the eighth character of each name that is eight or more characters long in the source file.

NAME

`crypt` - encode/decode

SYNOPSIS

`crypt` [`password`]

DESCRIPTION

Crypt reads from the standard input and writes on the standard output. The *password* is a key that selects a particular transformation. If no *password* is given, *crypt* demands a key from the terminal and turns off printing while the key is being typed in. *Crypt* encrypts and decrypts with the same key:

`crypt key <clear >cypher`

`crypt key <cypher | pr`

will print the clear.

Files encrypted by *crypt* are compatible with those treated by the editor *ed* in encryption mode.

The security of encrypted files depends on three factors: the fundamental method must be hard to solve; direct search of the key space must be infeasible; 'sneak paths' by which keys or cleartext can become visible must be minimized.

Crypt implements a one-rotor machine designed along the lines of the German Enigma, but with a 256-element rotor. Methods of attack on such machines are known, but not widely; moreover the amount of work required is likely to be large.

The transformation of a key into the internal settings of the machine is deliberately designed to be expensive, i.e. to take a substantial fraction of a second to compute. However, if keys are restricted to (say) three lower-case letters, then encrypted files can be read by expending only a substantial fraction of five minutes of machine time.

Since the key is an argument to the *crypt* command, it is potentially visible to users executing *ps*(1) or a derivative. To minimize this possibility, *crypt* takes care to destroy any record of the key immediately upon entry. No doubt the choice of keys and key security are the most vulnerable aspect of *crypt*.

FILES

`/dev/tty` for typed key

SEE ALSO

ed(1), *makekey*(8)

BUGS

There is no warranty of merchantability nor any warranty of fitness for a particular purpose nor any other warranty, either express or implied, as to the accuracy of the enclosed materials or as to their suitability for any particular purpose. Accordingly, Bell Telephone Laboratories assumes no responsibility for their use by the recipient. Further, Bell Laboratories assumes no obligation to furnish any assistance of any kind whatsoever, or to furnish any additional information or documentation.

The first part of the document discusses the importance of maintaining accurate records of all transactions. It emphasizes that proper record-keeping is essential for the smooth operation of any business and for the protection of its interests. The document also mentions the need for regular audits and the importance of having a clear system of accounting.

In the second part, the document outlines the various methods used to collect and analyze data. It describes the process of gathering information from different sources and how this data is then used to make informed decisions. The document also discusses the importance of having a good understanding of the market and the needs of the customers.

The third part of the document focuses on the financial aspects of the business. It discusses the various ways in which the business can raise capital and how this capital is then used to fund its operations. The document also mentions the importance of having a good understanding of the financial statements and the need for regular financial reporting.

In the final part, the document discusses the various ways in which the business can expand its operations. It mentions the importance of having a good understanding of the market and the needs of the customers, and how this can be used to develop new products and services. The document also discusses the importance of having a good understanding of the financial statements and the need for regular financial reporting.

NAME

`cs`h - a shell (command interpreter) with C-like syntax

SYNOPSIS

`cs`h [`-cefinstvVxX`] [`arg ...`]

DESCRIPTION

Csh is a first implementation of a command language interpreter incorporating a history mechanism (see **History Substitutions**) and a C-like syntax.

An instance of *cs*h begins by executing commands from the file `.cshrc` in the *home* directory of the invoker. If this is a login shell then it also executes commands from the file `.login` there. It is typical for users on crt's to put the command `"stty crt"` in their `.login` file, and to also invoke `tabs(1)` there.

In the normal case, the shell will then begin reading commands from the terminal, prompting with `%` . Processing of arguments and the use of the shell to process files containing command scripts will be described later.

The shell then repeatedly performs the following actions: a line of command input is read and broken into *words*. This sequence of words is placed on the command history list and then parsed. Finally each command in the current line is executed.

When a login shell terminates it executes commands from the file `.logout` in the users home directory.

Lexical structure

The shell splits input lines into words at blanks and tabs with the following exceptions. The characters `&` `|` `;` `<` `>` `(` `)` form separate words. If doubled in `&&`, `||`, `<<` or `>>` these pairs form single words. These parser metacharacters may be made part of other words, or prevented their special meaning, by preceding them with `\`. A newline preceded by a `\` is equivalent to a blank.

In addition strings enclosed in matched pairs of quotations, `"`, `'` or `'''`, form parts of a word; metacharacters in these strings, including blanks and tabs, do not form separate words. These quotations have semantics to be described subsequently. Within pairs of `"` or `'''` characters a newline preceded by a `\` gives a true newline character.

When the shell's input is not a terminal, the character `#` introduces a comment which continues to the end of the input line. It is prevented this special meaning when preceded by `\` and in quotations using `"`, `'`, and `'''`.

Commands

A simple command is a sequence of words, the first of which specifies the command to be executed. A simple command or a sequence of simple commands separated by `|` characters forms a pipeline. The output of each command in a pipeline is connected to the input of the next. Sequences of pipelines may be separated by `;`, and are then executed sequentially. A sequence of pipelines may be executed without immediately waiting for it to terminate by following it with an `&`.

Any of the above may be placed in '(' ') to form a simple command (which may be a component of a pipeline, etc.) It is also possible to separate pipelines with '|' or '&&' indicating, as in the C language, that the second is to be executed only if the first fails or succeeds respectively. (See *Expressions*.)

Jobs

The shell associates a *job* with each pipeline. It keeps a table of current jobs, printed by the *jobs* command, and assigns them small integer numbers. When a job is started asynchronously with '&', the shell prints a line which looks like:

```
[1] 1234
```

indicating that the job which was started asynchronously was job number 1 and had one (top-level) process, whose process id was 1234.

A job being run in the background will stop if it tries to read from the terminal. Background jobs are normally allowed to produce output, but this can be disabled by giving the command "stty tostop". If you set this tty option, then background jobs will stop when they try to produce output like they do when they try to read input.

The shell maintains a notion of the current and previous jobs. In output pertaining to jobs, the current job is marked with a '+' and the previous job with a '-'. The abbreviation '%+' refers to the current job and '%-' refers to the previous job. For close analogy with the syntax of the *history* mechanism (described below), '%%' is also a synonym for the current job.

Status reporting

This shell learns immediately whenever a process changes state. It normally informs you whenever a job becomes blocked so that no further progress is possible, but only just before it prints a prompt. This is done so that it does not otherwise disturb your work. If, however, you set the shell variable *notify*, the shell will notify you immediately of changes of status in background jobs. There is also a shell command *notify* which marks a single process so that its status changes will be immediately reported. By default *notify* marks the current process; simply say 'notify' after starting a background job to mark it.

Substitutions

We now describe the various transformations the shell performs on the input in the order in which they occur.

History substitutions

History substitutions place words from previous command input as portions of new commands, making it easy to repeat commands, repeat arguments of a previous command in the current command, or fix spelling mistakes in the previous command with little typing and a high degree of confidence. History substitutions begin with the character '!' and may begin **anywhere** in the input stream (with the provision that they do not nest.) This '!' may be preceded by an '\' to prevent its special meaning; for convenience, a '!' is passed unchanged when it is followed by a blank, tab, newline, '=' or '('. (History substitutions also occur when an

input line begins with '^'. This special abbreviation will be described later.) Any input line which contains history substitution is echoed on the terminal before it is executed as it could have been typed without history substitution.

Commands input from the terminal which consist of one or more words are saved on the history list. The history substitutions reintroduce sequences of words from these saved commands into the input stream. The size of which is controlled by the *history* variable; the previous command is always retained, regardless of its value. Commands are numbered sequentially from 1.

For definiteness, consider the following output from the *history* command:

```

9 write michael
10 ex write.c
11 cat oldwrite.c
12 diff *write.c
```

The commands are shown with their event numbers. It is not usually necessary to use event numbers, but the current event number can be made part of the *prompt* by placing an '!' in the prompt string.

With the current event 13 we can refer to previous events by event number '!11', relatively as in '!-2' (referring to the same event), by a prefix of a command word as in '!d' for event 12 or '!wri' for event 9, or by a string contained in a word in the command as in '!?mic?' also referring to event 9. These forms, without further modification, simply reintroduce the words of the specified events, each separated by a single blank. As a special case '!!' refers to the previous command; thus '!!' alone is essentially a *redo*.

To select words from an event we can follow the event specification by a ':' and a designator for the desired words. The words of a input line are numbered from 0, the first (usually command) word being 0, the second word (first argument) being 1, etc. The basic word designators are:

```

0      first (command) word
n      n'th argument
↑      first argument, i.e. '1'
$      last argument
%      word matched by (immediately preceding) ?s? search
x-y    range of words
-y     abbreviates '0-y'
*      abbreviates '↑-$', or nothing if only 1 word in event
x*     abbreviates 'x-$'
x-     like 'x*' but omitting word '$'
```

The ':' separating the event specification from the word designator can be omitted if the argument selector begins with a '↑', '\$', '*', '-' or '%'. After the optional word designator can be placed a sequence of modifiers, each preceded by a ':'. The following modifiers are defined:

```

h      Remove a trailing pathname component, leaving the head.
r      Remove a trailing '.xxx' component, leaving the root name.
e      Remove all but the extension '.xxx' part.
```


`s/l/r` / Substitute `l` for `r`
`t` Remove all leading pathname components, leaving the tail.
`&` Repeat the previous substitution.
`g` Apply the change globally, prefixing the above, e.g. `'g&'`.
`p` Print the new command but do not execute it.
`q` Quote the substituted words, preventing further substitutions.
`x` Like `q`, but break into words at blanks, tabs and newlines.

Unless preceded by a `'g'` the modification is applied only to the first modifiable word. With substitutions, it is an error for no word to be applicable.

The left hand side of substitutions are not regular expressions in the sense of the editors, but rather strings. Any character may be used as the delimiter in place of `'/'`; a `'\"` quotes the delimiter into the `l` and `r` strings. The character `'&'` in the right hand side is replaced by the text from the left. A `'\"` quotes `'&'` also. A null `l` uses the previous string either from a `l` or from a contextual scan string `s` in `'!s?'`. The trailing delimiter in the substitution may be omitted if a newline follows immediately as may the trailing `'?'` in a contextual scan.

A history reference may be given without an event specification, e.g. `'!$'`. In this case the reference is to the previous command unless a previous history reference occurred on the same line in which case this form repeats the previous reference. Thus `'!$foo?↑ !$'` gives the first and last arguments from the command matching `'$foo?'`.

A special abbreviation of a history reference occurs when the first non-blank character of an input line is a `'↑'`. This is equivalent to `'!s↑'` providing a convenient shorthand for substitutions on the text of the previous line. Thus `'↑lb↑lib'` fixes the spelling of `'lib'` in the previous command. Finally, a history substitution may be surrounded with `'{'` and `'}'` if necessary to insulate it from the characters which follow. Thus, after `'ls -ld ~paul'` we might do `'!{l}a'` to do `'ls -ld ~paula'`, while `'!la'` would look for a command starting `'la'`.

Quotations with `'` and `"`

The quotation of strings by `'` and `"` can be used to prevent all or some of the remaining substitutions. Strings enclosed in `'` are prevented any further interpretation. Strings enclosed in `"` are yet variable and command expanded as described below.

In both cases the resulting text becomes (all or part of) a single word; only in one special case (see *Command Substitution* below) does a `"` quoted string yield parts of more than one word; `'` quoted strings never do.

Alias substitution

The shell maintains a list of aliases which can be established, displayed and modified by the *alias* and *unalias* commands. After a command line is scanned, it is parsed into distinct commands and the first word of each command, left-to-right, is checked to see if it has an alias. If it does, then the text which is the alias for that command is reread with the history mechanism available as though that command were the previous input line. The resulting words replace the command and argument list.

If no reference is made to the history list, then the argument list is left unchanged.

Thus if the alias for 'ls' is 'ls -l' the command 'ls /usr' would map to 'ls -l /usr', the argument list here being undisturbed. Similarly if the alias for 'lookup' was 'grep !f /etc/passwd' then 'lookup bill' would map to 'grep bill /etc/passwd'.

If an alias is found, the word transformation of the input text is performed and the aliasing process begins again on the reformed input line. Looping is prevented if the first word of the new text is the same as the old by flagging it to prevent further aliasing. Other loops are detected and cause an error.

Note that the mechanism allows aliases to introduce parser metasyntax. Thus we can 'alias print 'pr \!* | lpr'' to make a command which *pr*'s its arguments to the line printer.

Variable substitution

The shell maintains a set of variables, each of which has as value a list of zero or more words. Some of these variables are set by the shell or referred to by it. For instance, the *argv* variable is an image of the shell's argument list, and words of this variable's value are referred to in special ways.

The values of variables may be displayed and changed by using the *set* and *unset* commands. Of the variables referred to by the shell a number are toggles; the shell does not care what their value is, only whether they are set or not. For instance, the *verbose* variable is a toggle which causes command input to be echoed. The setting of this variable results from the *-v* command line option.

Other operations treat variables numerically. The '@' command permits numeric calculations to be performed and the result assigned to a variable. Variable values are, however, always represented as (zero or more) strings. For the purposes of numeric operations, the null string is considered to be zero, and the second and subsequent words of multiword values are ignored.

After the input line is aliased and parsed, and before each command is executed, variable substitution is performed keyed by '\$' characters. This expansion can be prevented by preceding the '\$' with a '\' except within ""'s where it **always** occurs, and within "'s where it **never** occurs. Strings quoted by '"' are interpreted later (see *Command substitution* below) so '\$' substitution does not occur there until later, if at all. A '\$' is passed unchanged if followed by a blank, tab, or end-of-line.

Input/output redirections are recognized before variable expansion, and are variable expanded separately. Otherwise, the command name and entire argument list are expanded together. It is thus possible for the first (command) word to this point to generate more than one word, the first of which becomes the command name, and the rest of which become arguments.

Unless enclosed in "" or given the ':q' modifier the results of variable substitution may eventually be command and filename substituted. Within "" a variable whose value consists of multiple words expands to a

(portion of) a single word, with the words of the variables value separated by blanks. When the 'q' modifier is applied to a substitution the variable will expand to multiple words with each word separated by a blank and quoted to prevent later command or filename substitution.

The following metasequences are provided for introducing variable values into the shell input. Except as noted, it is an error to reference a variable which is not set.

`$name`
`${name}`

Are replaced by the words of the value of variable *name*, each separated by a blank. Braces insulate *name* from following characters which would otherwise be part of it. Shell variables have names consisting of up to 20 letters and digits starting with a letter. The underscore character is considered a letter.

If *name* is not a shell variable, but is set in the environment, then that value is returned (but `:` modifiers and the other forms given below are not available in this case).

`$name[selector]`
`${name[selector]}`

May be used to select only some of the words from the value of *name*. The selector is subjected to '\$' substitution and may consist of a single number or two numbers separated by a '-'. The first word of a variables value is numbered '1'. If the first number of a range is omitted it defaults to '1'. If the last member of a range is omitted it defaults to '\$#name'. The selector '*' selects all words. It is not an error for a range to be empty if the second argument is omitted or in range.

`$#name`
`${#name}`

Gives the number of words in the variable. This is useful for later use in a '[selector]'.

`$0`

Substitutes the name of the file from which command input is being read. An error occurs if the name is not known.

`$number`
`${number}`

Equivalent to '\$argv[number]'.

`$*`

Equivalent to '\$argv[*]'.

The modifiers 'h', 't', 'r', 'q' and 'x' may be applied to the substitutions above as may 'gh', 'gt' and 'gr'. If braces '{' '}' appear in the command form then the modifiers must appear within the braces. **The current implementation allows only one ':' modifier on each '\$' expansion.**

The following substitutions may not be modified with ':' modifiers.

`$?name`
`${?name}`

Substitutes the string '1' if name is set, '0' if it is not.

\$?

Substitutes '1' if the current input filename is known, '0' if it is not.

\$\$

Substitute the (decimal) process number of the (parent) shell.

\$<

Substitutes a line from the standard input, with no further interpretation thereafter. It can be used to read from the keyboard in a shell script.

Command and filename substitution

The remaining substitutions, command and filename substitution, are applied selectively to the arguments of builtin commands. This means that portions of expressions which are not evaluated are not subjected to these expansions. For commands which are not internal to the shell, the command name is substituted separately from the argument list. This occurs very late, after input-output redirection is performed, and in a child of the main shell.

Command substitution

Command substitution is indicated by a command enclosed in ``'. The output from such a command is normally broken into separate words at blanks, tabs and newlines, with null words being discarded, this text then replacing the original string. Within ``'s, only newlines force new words; blanks and tabs are preserved.

In any case, the single final newline does not force a new word. Note that it is thus possible for a command substitution to yield only part of a word, even if the command outputs a complete line.

Filename substitution

If a word contains any of the characters '*', '?', '[' or '{' or begins with the character '~', then that word is a candidate for filename substitution, also known as 'globbing'. This word is then regarded as a pattern, and replaced with an alphabetically sorted list of file names which match the pattern. In a list of words specifying filename substitution it is an error for no pattern to match an existing file name, but it is not required for each pattern to match. Only the metacharacters '*', '?' and '[' imply pattern matching, the characters '~' and '{' being more akin to abbreviations.

In matching filenames, the character '.' at the beginning of a filename or immediately following a '/', as well as the character '/' must be matched explicitly. The character '*' matches any string of characters, including the null string. The character '?' matches any single character. The sequence '['...' matches any one of the characters enclosed. Within '['...', a pair of characters separated by '-' matches any character lexically between the two.

The character '~' at the beginning of a filename is used to refer to home directories. Standing alone, i.e. '~' it expands to the invokers home directory as reflected in the value of the variable *home*. When followed by a name consisting of letters, digits and '-' characters the shell searches for a user with that name and substitutes their home directory; thus

'~ken' might expand to '/usr/ken' and '~ken/chmach' to '/usr/ken/chmach'. If the character '~' is followed by a character other than a letter or '/' or appears not at the beginning of a word, it is left undisturbed.

The metanotation 'a{b,c,d}e' is a shorthand for 'abe ace ade'. Left to right order is preserved, with results of matches being sorted separately at a low level to preserve this order. This construct may be nested. Thus '~source/s1/{oldls,ls}.c' expands to '/usr/source/s1/oldls.c /usr/source/s1/ls.c' whether or not these files exist without any chance of error if the home directory for 'source' is '/usr/source'. Similarly './{memo,*box}' might expand to './memo ../box ../mbox'. (Note that 'memo' was not sorted with the results of matching '*box'.) As a special case '{', '}' and '{} are passed undisturbed.

Input/output

The standard input and standard output of a command may be redirected with the following syntax:

< name

Open file *name* (which is first variable, command and filename expanded) as the standard input.

<< word

Read the shell input up to a line which is identical to *word*. *Word* is not subjected to variable, filename or command substitution, and each input line is compared to *word* before any substitutions are done on this input line. Unless a quoting '\', '"', "'" or '"' appears in *word* variable and command substitution is performed on the intervening lines, allowing '\' to quote '\$', '\' and '"'. Commands which are substituted have all blanks, tabs, and newlines preserved, except for the final newline which is dropped. The resultant text is placed in an anonymous temporary file which is given to the command as standard input.

> name

>! name

>& name

>&! name

The file *name* is used as standard output. If the file does not exist then it is created; if the file exists, it is truncated, its previous contents being lost.

If the variable *noclobber* is set, then the file must not exist or be a character special file (e.g. a terminal or '/dev/null') or an error results. This helps prevent accidental destruction of files. In this case the '!' forms can be used and suppress this check.

The forms involving '&' route the diagnostic output into the specified file as well as the standard output. *Name* is expanded in the same way as '<' input filenames are.

>> name

>>& name

>>! name

>>&! name

Uses file *name* as standard output like '>' but places output at the end of the file. If the variable *noclobber* is set, then it is an error for the file not to exist unless one of the '!' forms is given. Otherwise similar to '>'.

A command receives the environment in which the shell was invoked as modified by the input-output parameters and the presence of the command in a pipeline. Thus, unlike some previous shells, commands run from a file of shell commands have no access to the text of the commands by default; rather they receive the original standard input of the shell. The '<<' mechanism should be used to present inline data. This permits shell command scripts to function as components of pipelines and allows the shell to block read its input. Note that the default standard input for a command run detached is **not** modified to be the empty file '/dev/null'; rather the standard input remains as the original standard input of the shell. If this is a terminal and if the process attempts to read from the terminal, then the process will block and the user will be notified (see **Jobs** above.)

Diagnostic output may be directed through a pipe with the standard output. Simply use the form '|&' rather than just '|'.

Expressions

A number of the builtin commands (to be described subsequently) take expressions, in which the operators are similar to those of C, with the same precedence. These expressions appear in the @, *exit*, *if*, and *while* commands. The following operators are available:

|| && | ↑ & == != =~ !~ <= >= < > << >> + - * / % ! ~
()

Here the precedence increases to the right, '==' '!=' ' =~ ' and '!~', '<=' '>=' '<' and '>', '<<' and '>>', '+' and '-', '*' '/' and '%' being, in groups, at the same level. The '==' '!=' ' =~ ' and '!~' operators compare their arguments as strings; all others operate on numbers. The operators ' =~ ' and '!~' are like '!=' and '==' except that the right hand side is a *pattern* (containing, e.g. '*'s, '?'s and instances of '[...]') against which the left hand operand is matched. This reduces the need for use of the *switch* statement in shell scripts when all that is really needed is pattern matching.

Strings which begin with '0' are considered octal numbers. Null or missing arguments are considered '0'. The result of all expressions are strings, which represent decimal numbers. It is important to note that no two components of an expression can appear in the same word; except when adjacent to components of expressions which are syntactically significant to the parser ('&' '|' '<' '>' '(' ')') they should be surrounded by spaces.

Also available in expressions as primitive operands are command executions enclosed in '{' and '}' and file enquiries of the form '-l name' where *l* is one of:

r	read access
w	write access

x	execute access
e	existence
o	ownership
z	zero size
f	plain file
d	directory

The specified name is command and filename expanded and then tested to see if it has the specified relationship to the real user. If the file does not exist or is inaccessible then all enquiries return false, i.e. '0'. Command executions succeed, returning true, i.e. '1', if the command exits with status 0, otherwise they fail, returning false, i.e. '0'. If more detailed status information is required then the command should be executed outside of an expression and the variable *status* examined.

Control flow

The shell contains a number of commands which can be used to regulate the flow of control in command files (shell scripts) and (in limited but useful ways) from terminal input. These commands all operate by forcing the shell to reread or skip in its input and, due to the implementation, restrict the placement of some of the commands.

The *foreach*, *switch*, and *while* statements, as well as the *if-then-else* form of the *if* statement require that the major keywords appear in a single simple command on an input line as shown below.

If the shell's input is not seekable, the shell buffers up input whenever a loop is being read and performs seeks in this internal buffer to accomplish the rereading implied by the loop. (To the extent that this allows, backward goto's will succeed on non-seekable inputs.)

Builtin commands

Builtin commands are executed within the shell. If a builtin command occurs as any component of a pipeline except the last then it is executed in a subshell.

alias

alias name

alias name wordlist

The first form prints all aliases. The second form prints the alias for name. The final form assigns the specified *wordlist* as the alias of *name*; *wordlist* is command and filename substituted. *Name* is not allowed to be *alias* or *unalias*.

break

Causes execution to resume after the *end* of the nearest enclosing *foreach* or *while*. The remaining commands on the current line are executed. Multi-level breaks are thus possible by writing them all on one line.

breaksw

Causes a break from a *switch*, resuming after the *endsw*.

case label:

A label in a *switch* statement as discussed below.

cd

cd name

chdir

chdir name

Change the shells working directory to directory *name*. If no argument is given then change to the home directory of the user.

If *name* is not found as a subdirectory of the current directory (and does not begin with '/', './' or '../'), then each component of the variable *cdpath* is checked to see if it has a subdirectory *name*.

Finally, if all else fails but *name* is a shell variable whose value begins with '/', then this is tried to see if it is a directory.

continue

Continue execution of the nearest enclosing *while* or *foreach*. The rest of the commands on the current line are executed.

default:

Labels the default case in a *switch* statement. The default should come after all *case* labels.

dirs

Prints the directory stack; the top of the stack is at the left, the first directory in the stack being the current directory.

echo wordlist

echo -n wordlist

The specified words are written to the shells standard output, separated by spaces, and terminated with a newline unless the *-n* option is specified.

else

end

endif

endsw

See the description of the *foreach*, *if*, *switch*, and *while* statements below.

eval arg ...

(As in *sh*(1).) The arguments are read as input to the shell and the resulting command(s) executed. This is usually used to execute commands generated as the result of command or variable substitution, since parsing occurs before these substitutions. See *tset*(1) for an example of using *eval*.

exec command

The specified command is executed in place of the current shell.

exit

exit(expr)

The shell exits either with the value of the *status* variable (first form) or with the value of the specified *expr* (second form).

foreach name (wordlist)

...

end

The variable *name* is successively set to each member of *wordlist* and the sequence of commands between this command and the

matching *end* are executed. (Both *foreach* and *end* must appear alone on separate lines.)

The builtin command *continue* may be used to continue the loop prematurely and the builtin command *break* to terminate it prematurely. When this command is read from the terminal, the loop is read up once prompting with '?' before any statements in the loop are executed. If you make a mistake typing in a loop at the terminal you can rub it out.

glob wordlist

Like *echo* but no '\ ' escapes are recognized and words are delimited by null characters in the output. Useful for programs which wish to use the shell to filename expand a list of words.

goto word

The specified *word* is filename and command expanded to yield a string of the form 'label'. The shell rewinds its input as much as possible and searches for a line of the form 'label:' possibly preceded by blanks or tabs. Execution continues after the specified line.

hashstat

Print a statistics line indicating how effective the internal hash table has been at locating commands (and avoiding *exec*'s). An *exec* is attempted for each component of the *path* where the hash function indicates a possible hit, and in each component which does not begin with a '/ '.

history

history *n*

history -r *n*

history -h *n*

Displays the history event list; if *n* is given only the *n* most recent events are printed. The -r option reverses the order of printout to be most recent first rather than oldest first. The -h option causes the history list to be printed without leading numbers. This is used to produce files suitable for sourcing using the -h option to *source*.

if (expr) command

If the specified expression evaluates true, then the single *command* with arguments is executed. Variable substitution on *command* happens early, at the same time it does for the rest of the *if* command. *Command* must be a simple command, not a pipeline, a command list, or a parenthesized command list. Input/output redirection occurs even if *expr* is false, when command is not executed (this is a bug).

if (expr) then

...

else if (expr2) then

...

else

...

endif

If the specified *expr* is true then the commands to the first *else* are executed; else if *expr2* is true then the commands to the second *else*

are executed, etc. Any number of *else-if* pairs are possible; only one *endif* is needed. The *else* part is likewise optional. (The words *else* and *endif* must appear at the beginning of input lines; the *if* must appear alone on its input line or after an *else*.)

jobs

jobs -l

Lists the active jobs; given the *-l* options lists process id's in addition to the normal information.

kill %job

kill -sig %job ...

kill pid

kill -sig pid ...

kill -l

Sends either the TERM (terminate) signal or the specified signal to the specified jobs or processes. Signals are either given by number or by names (as given in */usr/include/signal.h*, stripped of the prefix "SIG"). The signal names are listed by "kill -l". There is no default, saying just 'kill' does not send a signal to the current job.

limit

limit resource

limit resource maximum-use

Limits the consumption by the current process and each process it creates to not individually exceed *maximum-use* on the specified *resource*. If no *maximum-use* is given, then the current limit is printed; if no *resource* is given, then all limitations are given.

Resources controllable currently include *cputime* (the maximum number of cpu-seconds to be used by each process), *filesize* (the largest single file which can be created), *datasize* (the maximum growth of the data+stack region via *sbrk(2)* beyond the end of the program text), *stacksize* (the maximum size of the automatically-extended stack region), and *coredumpsize* (the size of the largest core dump that will be created).

The *maximum-use* may be given as a (floating point or integer) number followed by a scale factor. For all limits other than *cputime* the default scale is 'k' or 'kilobytes' (1024 bytes); a scale factor of 'm' or 'megabytes' may also be used. For *cputime* the default scaling is 'seconds', while 'm' for minutes or 'h' for hours, or a time of the form 'mm:ss' giving minutes and seconds may be used.

For both *resource* names and scale factors, unambiguous prefixes of the names suffice.

login

Terminate a login shell, replacing it with an instance of */bin/login*. This is one way to log off, included for compatibility with *sh(1)*.

logout

Terminate a login shell. Especially useful if *ignoreeof* is set.

nice

nice +number

nice command**nice +number command**

The first form sets the *nice* for this shell to 4. The second form sets the *nice* to the given number. The final two forms run command at priority 4 and *number* respectively. The super-user may specify negative niceness by using '*nice -number ...*'. Command is always executed in a sub-shell, and the restrictions place on commands in simple *if* statements apply.

nohup**nohup command**

The first form can be used in shell scripts to cause hangups to be ignored for the remainder of the script. The second form causes the specified command to be run with hangups ignored. All processes detached with '&' are effectively *nohup*'ed.

notify**notify %job ...**

Causes the shell to notify the user asynchronously when the status of the current or specified jobs changes; normally notification is presented before a prompt. This is automatic if the shell variable *notify* is set.

onintr**onintr -****onintr label**

Control the action of the shell on interrupts. The first form restores the default action of the shell on interrupts which is to terminate shell scripts or to return to the terminal command input level. The second form '*onintr -*' causes all interrupts to be ignored. The final form causes the shell to execute a '*goto label*' when an interrupt is received or a child process terminates because it was interrupted.

In any case, if the shell is running detached and interrupts are being ignored, all forms of *onintr* have no meaning and interrupts continue to be ignored by the shell and all invoked commands.

popd**popd +n**

Pops the directory stack, returning to the new top directory. With a argument '*+n*' discards the *n*th entry in the stack. The elements of the directory stack are numbered from 0 starting at the top.

pushd**pushd name****pushd +n**

With no arguments, *pushd* exchanges the top two elements of the directory stack. Given a *name* argument, *pushd* changes to the new directory (ala *cd*) and pushes the old current working directory (as in *csw*) onto the directory stack. With a numeric argument, rotates the *n*th argument of the directory stack around to be the top element and changes to it. The members of the directory stack are numbered from the top starting at 0.

rehash

Causes the internal hash table of the contents of the directories in

the *path* variable to be recomputed. This is needed if new commands are added to directories in the *path* while you are logged in. This should only be necessary if you add commands to one of your own directories, or if a systems programmer changes the contents of one of the system directories.

repeat count command

The specified *command* which is subject to the same restrictions as the *command* in the one line *if* statement above, is executed *count* times. I/O redirections occur exactly once, even if *count* is 0.

set

set name

set name=word

set name[index]=word

set name=(wordlist)

The first form of the command shows the value of all shell variables. Variables which have other than a single word as value print as a parenthesized word list. The second form sets *name* to the null string. The third form sets *name* to the single *word*. The fourth form sets the *index**th* component of *name* to *word*; this component must already exist. The final form sets *name* to the list of words in *wordlist*. In all cases the value is command and filename expanded.

These arguments may be repeated to set multiple values in a single set command. Note however, that variable expansion happens for all arguments before any setting occurs.

setenv name value

Sets the value of environment variable *name* to be *value*, a single string. The most commonly used environment variable USER, TERM, and PATH are automatically imported to and exported from the *cs*h variables *user*, *term*, and *path*; there is no need to use *setenv* for these.

shift

shift variable

The members of *argv* are shifted to the left, discarding *argv*[1]. It is an error for *argv* not to be set or to have less than one word as value. The second form performs the same function on the specified variable.

source name

source -h name

The shell reads commands from *name*. *Source* commands may be nested; if they are nested too deeply the shell may run out of file descriptors. An error in a *source* at any level terminates all nested *source* commands. Normally input during *source* commands is not placed on the history list; the -h option causes the commands to be placed in the history list without being executed.

switch (string)

case str1:

...

breaksw

...
default:

...
breaksw
endsw

Each case label is successively matched, against the specified *string* which is first command and filename expanded. The file metacharacters '*', '?' and '['...' may be used in the case labels, which are variable expanded. If none of the labels match before a 'default' label is found, then the execution begins after the default label. Each case label and the default label must appear at the beginning of a line. The command *breaksw* causes execution to continue after the *endsw*. Otherwise control may fall through case labels and default labels as in C. If no label matches and there is no default, execution continues after the *endsw*.

time
time command

With no argument, a summary of time used by this shell and its children is printed. If arguments are given the specified simple command is timed and a time summary as described under the *time* variable is printed. If necessary, an extra shell is created to print the time statistic when the command completes.

umask
umask value

The file creation mask is displayed (first form) or set to the specified value (second form). The mask is given in octal. Common values for the mask are 002 giving all access to the group and read and execute access to others or 022 giving all access except no write access for users in the group or others.

unalias pattern

All aliases whose names match the specified pattern are discarded. Thus all aliases are removed by 'unalias *'. It is not an error for nothing to be *unaliased*.

unhash

Use of the internal hash table to speed location of executed programs is disabled.

unlimit resource
unlimit

Removes the limitation on *resource*. If no *resource* is specified, then all *resource* limitations are removed.

unset pattern

All variables whose names match the specified pattern are removed. Thus all variables are removed by 'unset *'; this has noticeably distasteful side-effects. It is not an error for nothing to be *unset*.

unsetenv pattern

Removes all variables whose name match the specified pattern from the environment. See also the *setenv* command above and *printenv*(1).

wait

All background jobs are waited for. If the shell is interactive, then an interrupt can disrupt the wait, at which time the shell prints names and job numbers of all jobs known to be outstanding.

while (expr)

...
end

While the specified expression evaluates non-zero, the commands between the *while* and the matching end are evaluated. *Break* and *continue* may be used to terminate or continue the loop prematurely. (The *while* and *end* must appear alone on their input lines.) Prompting occurs here the first time through the loop as for the *foreach* statement if the input is a terminal.

@

@ name = expr

@ name[index] = expr

The first form prints the values of all the shell variables. The second form sets the specified *name* to the value of *expr*. If the expression contains '<', '>', '&' or '|' then at least this part of the expression must be placed within '(' ')'. The third form assigns the value of *expr* to the *index**th* argument of *name*. Both *name* and its *index**th* component must already exist.

The operators '*=', '+=', etc are available as in C. The space separating the name from the assignment operator is optional. Spaces are, however, mandatory in separating components of *expr* which would otherwise be single words.

Special postfix '++' and '--' operators increment and decrement *name* respectively, i.e. '@ i++'.

Pre-defined and environment variables

The following variables have special meaning to the shell. Of these, *argv*, *cwd*, *home*, *path*, *prompt*, *shell* and *status* are always set by the shell. Except for *cwd* and *status* this setting occurs only at initialization; these variables will not then be modified unless this is done explicitly by the user.

This shell copies the environment variable USER into the variable *user*, TERM into *term*, and HOME into *home*, and copies these back into the environment whenever the normal shell variables are reset. The environment variable PATH is likewise handled; it is not necessary to worry about its setting other than in the file *.cshrc* as inferior *csh* processes will import the definition of *path* from the environment, and re-export it if you then change it. (It could be set once in the *.login* except that commands through *net(1)* would not see the definition.)

argv Set to the arguments to the shell, it is from this variable that positional parameters are substituted, i.e. '\$1' is replaced by '\$argv[1]', etc.

cdpath Gives a list of alternate directories searched to find sub-directories in *chdir* commands.

pwd	The full pathname of the current directory.
echo	Set when the -x command line option is given. Causes each command and its arguments to be echoed just before it is executed. For non-builtin commands all expansions occur before echoing. Builtin commands are echoed before command and filename substitution, since these substitutions are then done selectively.
histchars	Can be given a string value to change the characters used in history substitution. The first character of its value is used as the history substitution character, replacing the default character ! . The second character of its value replaces the character ↑ in quick substitutions.
history	Can be given a numeric value to control the size of the history list. Any command which has been referenced in this many events will not be discarded. Too large values of <i>history</i> may run the shell out of memory. The last executed command is always saved on the history list.
home	The home directory of the invoker, initialized from the environment. The filename expansion of ~ refers to this variable.
ignoreeof	If set the shell ignores end-of-file from input devices which are terminals. This prevents shells from accidentally being killed by control-D's.
mail	<p>The files where the shell checks for mail. This is done after each command completion which will result in a prompt, if a specified interval has elapsed. The shell says 'You have new mail.' if the file exists with an access time not greater than its modify time.</p> <p>If the first word of the value of <i>mail</i> is numeric it specifies a different mail checking interval, in seconds, than the default, which is 10 minutes.</p> <p>If multiple mail files are specified, then the shell says 'New mail in <i>name</i>' when there is mail in the file <i>name</i>.</p>
noclobber	As described in the section on <i>Input/output</i> , restrictions are placed on output redirection to insure that files are not accidentally destroyed, and that >> redirections refer to existing files.
noglob	If set, filename expansion is inhibited. This is most useful in shell scripts which are not dealing with filenames, or after a list of filenames has been obtained and further expansions are not desirable.
nonomatch	If set, it is not an error for a filename expansion to not match any existing files; rather the primitive pattern is returned. It is still an error for the primitive pattern to be malformed, i.e. 'echo [' still gives an error.
notify	If set, the shell notifies asynchronously of job completions. The default is to rather present job completions just before

printing a prompt.

path

Each word of the *path* variable specifies a directory in which commands are to be sought for execution. A null word specifies the current directory. If there is no *path* variable then only full path names will execute. The usual search path is '.', '/bin' and '/usr/bin', but this may vary from system to system. For the super-user the default search path is '/etc', '/bin' and '/usr/bin'. A shell which is given neither the *-c* nor the *-t* option will normally hash the contents of the directories in the *path* variable after reading *.cshrc*, and each time the *path* variable is reset. If new commands are added to these directories while the shell is active, it may be necessary to give the *rehash* or the commands may not be found.

prompt

The string which is printed before each command is read from an interactive terminal input. If a '*!*' appears in the string it will be replaced by the current event number unless a preceding '**' is given. Default is '%', or '#' for the super-user.

savehist

is given a numeric value to control the number of entries of the history list that are saved in *~/.history* when the user logs out. Any command which has been referenced in this many events will be saved. During start up the shell sources *~/.history* into the history list enabling history to be saved across logins. Too large values of *savehist* will slow down the shell during start up.

shell

The file in which the shell resides. This is used in forking shells to interpret files which have execute bits set, but which are not executable by the system. (See the description of *Non-builtin Command Execution* below.) Initialized to the (system-dependent) home of the shell.

status

The status returned by the last command. If it terminated abnormally, then 0200 is added to the status. Builtin commands which fail return exit status '1', all other builtin commands set status '0'.

time

Controls automatic timing of commands. If set, then any command which takes more than this many cpu seconds will cause a line giving user, system, and real times and a utilization percentage which is the ratio of user plus system times to real time to be printed when it terminates.

verbose

Set by the *-v* command line option, causes the words of each command to be printed after history substitution.

Non-builtin command execution

When a command to be executed is found to not be a builtin command the shell attempts to execute the command via *exec(2)*. Each word in the variable *path* names a directory from which the shell will attempt to execute the command. If it is given neither a *-c* nor a *-t* option, the shell will hash the names in these directories into an internal table so that it

will only try an *exec* in a directory if there is a possibility that the command resides there. This greatly speeds command location when a large number of directories are present in the search path. If this mechanism has been turned off (via *unhash*), or if the shell was given a *-c* or *-t* argument, and in any case for each directory component of *path* which does not begin with a '/', the shell concatenates with the given command name to form a path name of a file which it then attempts to execute.

Parenthesized commands are always executed in a subshell. Thus '(cd ; pwd) ; pwd' prints the *home* directory; leaving you where you were (printing this after the *home* directory), while 'cd ; pwd' leaves you in the *home* directory. Parenthesized commands are most often used to prevent *chdir* from affecting the current shell.

If the file has execute permissions but is not an executable binary to the system, then it is assumed to be a file containing shell commands and a new shell is spawned to read it.

If there is an *alias* for *shell* then the words of the alias will be prepended to the argument list to form the shell command. The first word of the *alias* should be the full path name of the shell (e.g. '\$shell'). Note that this is a special, late occurring, case of *alias* substitution, and only allows words to be prepended to the argument list without modification.

Argument list processing

If argument 0 to the shell is '-' then this is a login shell. The flag arguments are interpreted as follows:

- c Commands are read from the (single) following argument which must be present. Any remaining arguments are placed in *argv*.
- e The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.
- f The shell will start faster, because it will neither search for nor execute commands from the file '.cshrc' in the invokers home directory.
- i The shell is interactive and prompts for its top-level input, even if it appears to not be a terminal. Shells are interactive without this option if their inputs and outputs are terminals.
- n Commands are parsed, but not executed. This may aid in syntactic checking of shell scripts.
- s Command input is taken from the standard input.
- t A single line of input is read and executed. A '\' may be used to escape the newline at the end of this line and continue onto another line.
- v Causes the *verbose* variable to be set, with the effect that command input is echoed after history substitution.
- x Causes the *echo* variable to be set, so that commands are echoed immediately before execution.
- V Causes the *verbose* variable to be set even before '.cshrc' is executed.

-X Is to **-x** as **-V** is to **-v**.

After processing of flag arguments if arguments remain but none of the **-c**, **-i**, **-s**, or **-t** options was given the first argument is taken as the name of a file of commands to be executed. The shell opens this file, and saves its name for possible resubstitution by '\$0'. Since many systems use either the standard version 6 or version 7 shells whose shell scripts are not compatible with this shell, the shell will execute such a 'standard' shell if the first character of a script is not a '#', i.e. if the script does not start with a comment. Remaining arguments initialize the variable *argv*.

Signal handling

The shell normally ignores *quit* signals. Jobs running detached (ie. in the background) are immune to signals generated from the keyboard, including hangups. Other signals have the values which the shell inherited from its parent. The shells handling of interrupts and terminate signals in shell scripts can be controlled by *onintr*. Login shells catch the *terminate* signal; otherwise this signal is passed on to children from the state in the shell's parent. In no case are interrupts allowed when a login shell is reading the file '.logout'.

AUTHOR

William Joy. Job control and directory stack features first implemented by J.E. Kulp of I.I.A.S.A, Laxenburg, Austria, with different syntax than that used now.

FILES

~/cshrc	Read at beginning of execution by each shell.
~/login	Read by login shell, after '.cshrc' at login.
~/logout	Read by login shell, at logout.
/bin/sh	Standard shell, for shell scripts not starting with a '#'
/tmp/sh*	Temporary file for '<<'
/etc/passwd	Source of home directories for '~name'.

LIMITATIONS

Words can be no longer than 1024 characters. The system limits argument lists to 10240 characters. The number of arguments to a command which involves filename expansion is limited to 1/6'th the number of characters allowed in an argument list. Command substitutions may substitute no more characters than are allowed in an argument list. To detect looping, the shell restricts the number of *alias* substitutions on a single line to 20.

SEE ALSO

sh(1), access(2), execve(2), fork(2), killpg(2), pipe(2), sigsys(2), umask(2), setrlimit(2), wait(2), intro(3J), sigset(3), tty(4), a.out(5), environ(5), 'An introduction to the C shell'

BUGS

The PCS *MUNIX* 1.5 port of the cshell has none of the job control primitives such as *bg*, *fg*, *stop*... This is because we do not use the Berkeley signals which are not in System V Unix.

When a command is restarted from a stop, the shell prints the directory it started in if this is different from the current directory; this can be misleading (i.e. wrong) as the job may have changed directories

internally.

Shell builtin functions are not stoppable/restartable. Command sequences of the form 'a ; b ; c' are also not handled gracefully when stopping is attempted. If you suspend 'b', the shell will then immediately execute 'c'. This is especially noticeable if this expansion results from an *alias*. It suffices to place the sequence of commands in ()'s to force it to a subshell, i.e. '(a ; b ; c)'.

Control over tty output after processes are started is primitive; perhaps this will inspire someone to work on a good virtual terminal interface. In a virtual terminal interface much more interesting things could be done with output control.

Alias substitution is most often used to clumsily simulate shell procedures; shell procedures should be provided rather than aliases.

Commands within loops, prompted for by '?', are not placed in the *history* list. Control structure should be parsed rather than being recognized as built-in commands. This would allow control commands to be placed anywhere, to be combined with '|', and to be used with '&' and ';' metasyntax.

It should be possible to use the ':' modifiers on the output of command substitutions. All and more than one ':' modifier should be allowed on '\$' substitutions.

Symbolic links fool the shell. In particular, *dirs* and 'cd ..' don't work properly once you've crossed through a symbolic link.

NAME

csplit - context split

SYNOPSIS

csplit [-s] [-k] [-f prefix] file arg1 [... argn]

DESCRIPTION

Csplit reads *file* and separates it into *n*+1 sections, defined by the arguments *arg1* ... *argn*. By default the sections are placed in *xx00* ... *xxn* (*n* may not be greater than 99). These sections get the following pieces of *file*:

- 00: From the start of *file* up to (but not including) the line referenced by *arg1*.
- 01: From the line referenced by *arg1* up to the line referenced by *arg2*.
- ...
- n*+1: From the line referenced by *argn* to the end of *file*.

The options to *csplit* are:

- s *Csplit* normally prints the character counts for each file created. If the -s option is present, *csplit* suppresses the printing of all character counts.
- k *Csplit* normally removes created files if an error occurs. If the -k option is present, *csplit* leaves previously created files intact.
- f *prefix* If the -f option is used, the created files are named *prefix00* ... *prefixn*. The default is *xx00* ... *xxn*.

The arguments (*arg1* ... *argn*) to *csplit* can be a combination of the following:

/regexp/

A file is to be created for the section from the current line up to (but not including) the line containing the regular expression *regexp*. The current line becomes the line containing *regexp*. This argument may be followed by an optional +*or* - some number of lines (e.g., */Page/-5*).

%*regexp*%

This argument is the same as */regexp/*, except that no file is created for the section.

lnno

A file is to be created from the current line up to (but not including) *lnno*. The current line becomes *lnno*.

{*num*}

Repeat argument. This argument may follow any of the above arguments. If it follows a *regexp* type argument, that argument is applied *num* more times. If it follows *lnno*, the file will be split every *lnno* lines (*num* times) from that point.

Enclose all *regexp* type arguments that contain blanks or other characters meaningful to the Shell in the appropriate quotes. Regular expressions may not contain embedded new-lines. *Csplit* does not affect the original file; it is the users responsibility to remove it.

EXAMPLES

```
csplit -f cobol file '/procedure division/' /par5./ /par16./
```

This example creates four files, **cobol00** ... **cobol03**. After editing the "split" files, they can be recombined as follows:

```
cat cobol0[0-3] > file
```

Note that this example overwrites the original file.

```
csplit -k file 100 {99}
```

This example would split the file at every 100 lines, up to 10,000 lines. The **-k** option causes the created files to be retained if there are less than 10,000 lines; however, an error message would still be printed.

```
csplit -k prog.c '%main(%)' '/~}/+1' {20}
```

Assuming that **prog.c** follows the normal C coding convention of ending routines with a **}** at the beginning of the line, this example will create a file containing each separate C routine (up to 21) in **prog.c**.

SEE ALSO

ed(1), **sh(1)**, **regexp(7)**.

DIAGNOSTICS

Self explanatory except for:

arg - out of range

which means that the given argument did not reference a line between the current position and the end of the file.

NAME

`ctags` - create a tags file

SYNOPSIS

`ctags` [`-u`] [`-v`] [`-w`] [`-x`] name ...

DESCRIPTION

`Ctags` makes a tags file for `ex(1)` from the specified C, Pascal and Fortran sources. A tags file gives the locations of specified objects (in this case functions) in a group of files. Each line of the tags file contains the function name, the file in which it is defined, and a scanning pattern used to find the function definition. These are given in separate fields on the line, separated by blanks or tabs. Using the `tags` file, `ex` can quickly find these function definitions.

If the `-x` flag is given, `ctags` produces a list of function names, the line number and file name on which each is defined, as well as the text of that line and prints this on the standard output. This is a simple index which can be printed out as an off-line readable function index.

If the `-v` flag is given, an index of the form expected by `vgrind(1)` is produced on the standard output. This listing contains the function name, file name, and page number (assuming 64 line pages). Since the output will be sorted into lexicographic order, it may be desired to run the output through `sort -f`. Sample use:

```
ctags -v files | sort -f > index
vgrind -x index
```

Files whose name ends in `.c` or `.h` are assumed to be C source files and are searched for C routine and macro definitions. Others are first examined to see if they contain any Pascal or Fortran routine definitions; if not, they are processed again looking for C definitions.

Other options are:

`-w` suppressing warning diagnostics.

`-u` causing the specified files to be *updated* in tags, that is, all references to them are deleted, and the new values are appended to the file. (Beware: this option is implemented in a way which is rather slow; it is usually faster to simply rebuild the `tags` file.)

The tag `main` is treated specially in C programs. The tag formed is created by prepending `M` to the name of the file, with a trailing `.c` removed, if any, and leading pathname components also removed. This makes use of `ctags` practical in directories with more than one program.

FILES

`tags` output tags file

SEE ALSO

`ex(1)`, `vi(1)`

AUTHOR

Ken Arnold; FORTRAN added by Jim Kleckner; Bill Joy added Pascal and `-x`, replacing `cxref`.

BUGS

Recognition of **functions**, **subroutines** and **procedures** for FORTRAN and Pascal is done in a very simpleminded way. No attempt is made to deal

with block structure; if you have two Pascal procedures in different blocks with the same name you lose.

The method of deciding whether to look for C or Pascal and FORTRAN functions is a hack.

NAME

cu - call UNIX

SYNOPSIS

cu telno [-t] [-s speed] [-a acu] [-l line] [-E eotc]

DESCRIPTION

Cu calls up another UNIX system, a terminal, or possibly a non-UNIX system. It manages an interactive conversation with possible transfers of text files. *Telno* is the telephone number, with minus signs at appropriate places for delays. The *-t* flag is used to dial out to a terminal. *Speed* gives the transmission speed (110, 134, 150, 300, 1200); 300 is the default value.

The *-a* and *-l* values may be used to specify pathnames for the ACU and communications line devices. They can be used to override the following built-in choices:

-a /dev/cua0 *-l* /dev/cul0

The *-E* option changes the EOT character from CTRL-Z to the (decimal) value *eotc*. For file transfer to a different UNIX system often the option *-E 4* must be given to set EOT to CTRL-D.

After making the connection, *cu* runs as two processes: the *send* process reads the standard input and passes most of it to the remote system; the *receive* process reads from the remote system and passes most data to the standard output. Lines beginning with '~' have special meanings.

The *send* process interprets the following:

- ~. terminate the conversation.
- ~EOT terminate the conversation
- ~<file send the contents of *file* to the remote system, as though typed at the terminal.
- ~! invoke an interactive shell on the local system.
- ~!cmd ... run the command on the local system (via *sh -c*).
- ~\$cmd ... run the command locally and send its output to the remote system.
- ~%take from [to] copy file 'from' (on the remote system) to file 'to' on the local system. If 'to' is omitted, the 'from' name is used both places.
- ~%put from [to] copy file 'from' (on local system) to file 'to' on remote system. If 'to' is omitted, the 'from' name is used both places.
- ~~ ... send the line '~...'

The *receive* process handles output diversions of the following form:

~>[>][:]file
zero or more lines to be written to file
~>

In any case, output is diverted (or appended, if '>>' used) to the file. If ':' is used, the diversion is *silent*, i.e., it is written only to the file. If '.' is

omitted, output is written both to the file and to the standard output. The trailing '~>' terminates the diversion.

The use of ~Xput requires stty and cat on the remote side. It also requires that the current erase and kill characters on the remote system be identical to the current ones on the local system. Backslashes are inserted at appropriate places.

The use of ~Xtake requires the existence of echo and tee on the remote system. Also, stty tabs mode is required on the remote system if tabs are to be copied without expansion.

EXAMPLE

```
cu -t -s 9600 -a /dev/null -l /dev/tty30
```

FILES

```
/dev/cua0  
/dev/cul0  
/dev/null
```

SEE ALSO

```
dn(4), tty(4)
```

DIAGNOSTICS

Exit code is zero for normal exit, nonzero (various values) otherwise.

BUGS

The syntax is unique.

NAME

`cut` - cut out selected fields of each line of a file

SYNOPSIS

```
cut -clist [file1 file2 ...]
cut -flist [-dchar] [-s] [file1 file2 ...]
```

DESCRIPTION

Use *cut* to cut out columns from a table or fields from each line of a file; in data base parlance, it implements the projection of a relation. The fields as specified by *list* can be fixed length, i.e., character positions as on a punched card (`-c` option), or the length can vary from line to line and be marked with a field delimiter character like *tab* (`-f` option). *Cut* can be used as a filter; if no files are given, the standard input is used.

The meanings of the options are:

- list* A comma-separated list of integer field numbers (in increasing order), with optional `-` to indicate ranges as in the `-o` option of *nroff*/*troff* for page ranges; e.g., 1,4,7; 1-3,8; -5,10 (short for 1-5,10); or 3- (short for third through last field).
- `-clist` The *list* following `-c` (no space) specifies character positions (e.g., `-c1-72` would pass the first 72 characters of each line).
- `-flist` The *list* following `-f` is a list of fields assumed to be separated in the file by a delimiter character (see `-d`); e.g., `-f1,7` copies the first and seventh field only. Lines with no field delimiters will be passed through intact (useful for table subheadings), unless `-s` is specified.
- `-dchar` The character following `-d` is the field delimiter (`-f` option only). Default is *tab*. Space or other characters with special meaning to the shell must be quoted.
- `-s` Suppresses lines with no delimiter characters in case of `-f` option. Unless specified, lines with no delimiters will be passed through untouched.

Either the `-c` or `-f` option must be specified.

HINTS

Use *grep*(1) to make horizontal "cuts" (by context) through a file, or *paste*(1) to put files together column-wise (i.e., horizontally). To reorder columns in a table, use *cut* and *paste*.

EXAMPLES

```
cut -d: -f1,5 /etc/passwd
                                mapping of user IDs to names
name=`who am i | cut -f1 -d" "`
                                to set name to current login name.
```

DIAGNOSTICS

line too long A line can have no more than 511 characters or fields.

bad list for c /f option

Missing `-c` or `-f` option or incorrectly specified *list*. No error occurs if a line has fewer fields than the *list* calls for.

CUT(1)

MUNIX

CUT(1)

no fields The *list* is empty.

SEE ALSO

grep(1), paste(1).

NAME

d - text database functions

DESCRIPTION

d is a set of functions that work on *Unix* text files with records' like:
 from: Smith to: Jones date: 15.5 re: Unix jobs

or:

Title: What is the Title of this Book?
 Author: R.M.Smullyan
 Year: 1978
 Subjects: logic, games, paradox

or Usenet mail, or a software project database like [Knudsen].

d is designed for small text databases like these, where

- the familiar, fast screen editor can edit the data
- *Unix* tools *sort*, *grep*, *awk*, *join* ... all work
- the size and cost of a big database system would be overkill.

d line, *d record*, *d get*, *d form*:

d has four subcommands:

d {line | record | get | form} [options] [files, or stdin]
line, *record*, *get*, *form* may be abbreviated *l*, *r*, *g*, *f*.

d line

makes each record into a single 'line', for *sort* or *awk* or ...

d record

undoes *d line* in:

d line afile | Unix-filter | d record

where *d line* flattens each record to one line,
 Unix-filter *sorts* or *awks* ... the lines, and
 d record makes each line back into a record.

d get aword

gets all records containing 'aword', like a fast
d line | fgrep aword | d record

d form aformfile

puts fields from the data base into a form, such as:

Dear {Name},
 In reply to your {Adjective} letter of {Date},
 ...

or simply selects some fields:

{Phone} {Name}.

EXAMPLES

d get Edinburgh eunet.map

looks in the *eunet* map for people in Edinburgh.

d line <project | sort '-t|'-r +2 | *d rec*
 sorts project records on field 3.

d record format

A record is a group of lines, followed by a blank line. A field is 'Fieldname: Value', where 'Value' is everything up to the next 'Fieldname': a word, a line, or several lines. 'Value' may contain a ':', but not right after a letter or number.

d line separates fields with a '|' field separator (*awk FS, sort -t*). *d record* then deletes all '|', so that *d line* | *d record* does nothing.

d line also changes newlines in multi-line records to an ersatz newline, default '\', which *d record* changes back to \n. To specify your own field-separator and newline characters (which must not occur in the data):

dsep='FN'; *export dsep* .
dsep='\ ' is the default.

C interface

d is only a few pages of C code, so it can be easily tailored or extended:

```
#include "field.h"
```

typedefs a *struct field*, which holds pointers to fieldname, fieldval etc.

```
nf = getFields(char*rec, field f[])
```

splits *rec* into 'fieldname: value' pairs, filling *f*[0 .. *nf*-1].

```
char* getPara(char nl)
```

returns the next paragraph (lines up to \n\n) from the open file *Paramfile*. The text is buffered inside *getPara*. Paragraphs may be no longer than 4096 bytes. \n s are changed to *nl*.

Possible extensions:

More complicated *gets*, such as *d get 'Duedate > 15.5'*.

Dates.

Iget, interactive *get*.

References

Knudsen D.B. et al, "A Modification Request Control System", in Proc. 2nd International Conference on Software Engineering, San Francisco, 1976, pp. 187-192.

Author

Denis Bzowy, PCS, (089) 67804-275, April 1983.

SEE ALSO

awk(1), cut(1), join(1), sort(1), dbm(3X), table(3-pcs)

BUGS

Sort, awk etc. may misbehave on records longer than 256 or 512 bytes.

100

100

100

NAME

date - print and set the date

SYNOPSIS

date [mmddhhmm[yy]] [+format]

DESCRIPTION

If no argument is given, or if the argument begins with +, the current date and time are printed. Otherwise, the current date is set. The first *mm* is the month number; *dd* is the day number in the month; *hh* is the hour number (24 hour system); the second *mm* is the minute number; *yy* is the last 2 digits of the year number and is optional. For example:

date 10080045

sets the date to Oct 8, 12:45 AM. The current year is the default if no year is mentioned. The system operates in GMT. *Date* takes care of the conversion to and from local standard and daylight time.

If the argument begins with +, the output of *date* is under the control of the user. The format for the output is similar to that of the first argument to *printf*(3S). All output fields are of fixed size (zero padded if necessary). Each field descriptor is preceded by % and will be replaced in the output by its corresponding value. A single % is encoded by %%. All other characters are copied to the output without change. The string is always terminated with a new-line character.

Field Descriptors:

n	insert a new-line character
t	insert a tab character
m	month of year - 01 to 12
d	day of month - 01 to 31
y	last 2 digits of year - 00 to 99
D	date as mm/dd/yy
H	hour - 00 to 23
M	minute - 00 to 59
S	second - 00 to 59
T	time as HH:MM:SS
j	day of year - 001 to 366
w	day of week - Sunday = 0
a	abbreviated weekday - Sun to Sat
h	abbreviated month - Jan to Dec
r	time in AM/PM notation

EXAMPLE

date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'
would have generated as output:
DATE: 08/01/76
TIME: 14:45:05

DIAGNOSTICS

<i>No permission</i>	if you aren't the super-user and you try to change the date;
<i>bad conversion</i>	if the date set is syntactically incorrect;
<i>bad format character</i>	if the field descriptor is not recognizable.

FILES

/dev/kmem

DATE(1)

MUNIX

DATE(1)

WARNING

It is a bad practice to change the date while the system is running multi-user.

NAME

dbadd - add entry to an Emacs data base
dbcreate - create an Emacs data base
dblist - list contents of an Emacs data base
dbprint - print an entry from an Emacs data base

SYNOPSIS

dbadd dbname key
dbcreate dbname
dblist dbname [-l] [-p newdbname]
dbprint dbname key

DESCRIPTION

All of these commands deal with databases used by the Unix Emacs database manipulation facilities. A Unix Emacs database is simply a set of (key,content) pairs, as in dbm(3X), except that the content part can be a very long string.

Dbadd adds the text from the standard input to the named database using the given key. **Dbcreate** creates the named database, making it empty. **Dbprint** prints the contents of the entry from the database with the given key.

Dblist with no arguments simply lists the keys of all the items in the database. With the -l option it prints some internal information from the database of no interest to anyone but the implementor. The -p option causes the key and content of every entry to be listed as a shell command file which when executed will repeatedly invoke **dbadd** to rebuild the database. This form of **dblist** is handy when you want a readable ascii file representation of a data base for shipping around or editing. Databases should be recreated periodically to garbage collect them.

FILES

dbname.dir, *dbname.pag*, and *dbname.dat*: the three component subfiles of a database.

SEE ALSO

James Gosling, *The Unix Emacs Manual*
emacs(1), dbm(3X) from which much code was stolen.

AUTHOR

James Gosling @ CMU

NAME

dc - desk calculator

SYNOPSIS

dc [file]

DESCRIPTION

Dc is an arbitrary precision arithmetic package. Ordinarily it operates on decimal integers, but one may specify an input base, output base, and a number of fractional digits to be maintained. The overall structure of dc is a stacking (reverse Polish) calculator. If an argument is given, input is taken from that file until its end, then from the standard input. The following constructions are recognized:

number

The value of the number is pushed on the stack. A number is an unbroken string of the digits 0-9. It may be preceded by an underscore _ to input a negative number. Numbers may contain decimal points.

+ - / * % ^

The top two values on the stack are added (+), subtracted (-), multiplied (*), divided (/), remaindered (%), or exponentiated (^). The two entries are popped off the stack; the result is pushed on the stack in their place. Any fractional part of an exponent is ignored.

sz The top of the stack is popped and stored into a register named *x*, where *x* may be any character. If the *s* is capitalized, *x* is treated as a stack and the value is pushed on it.

lx The value in register *x* is pushed on the stack. The register *x* is not altered. All registers start with zero value. If the *l* is capitalized, register *x* is treated as a stack and its top value is popped onto the main stack.

d The top value on the stack is duplicated.

p The top value on the stack is printed. The top value remains unchanged. **P** interprets the top of the stack as an ascii string, removes it, and prints it.

f All values on the stack and in registers are printed.

q exits the program. If executing a string, the recursion level is popped by two. If **q** is capitalized, the top value on the stack is popped and the string execution level is popped by that value.

x treats the top element of the stack as a character string and executes it as a string of dc commands.

X replaces the number on the top of the stack with its scale factor.

[...] puts the bracketed ascii string onto the top of the stack.

<x >x =x

The top two elements of the stack are popped and compared. Register *x* is executed if they obey the stated relation.

v replaces the top element on the stack by its square root. Any existing fractional part of the argument is taken into account, but otherwise the scale factor is ignored.

- ! interprets the rest of the line as a UNIX command.
- c All values on the stack are popped.
- i The top value on the stack is popped and used as the number radix for further input. I pushes the input base on the top of the stack.
- o The top value on the stack is popped and used as the number radix for further output.
- O pushes the output base on the top of the stack.
- k the top of the stack is popped, and that value is used as a non-negative scale factor: the appropriate number of places are printed on output, and maintained during multiplication, division, and exponentiation. The interaction of scale factor, input base, and output base will be reasonable if all are changed together.
- z The stack level is pushed onto the stack.
- Z replaces the number on the top of the stack with its length.
- ? A line of input is taken from the input source (usually the terminal) and executed.
- :: are used by *bc* for array operations.

An example which prints the first ten values of $n!$ is

```
[la1+dsa*pla10>y]sy
0sa1
lyx
```

SEE ALSO

bc(1), which is a preprocessor for *dc* providing infix notation and a C-like syntax which implements functions and reasonable control structures for programs.

DIAGNOSTICS

- 'x is unimplemented' where x is an octal number.
- 'stack empty' for not enough elements on the stack to do what was asked.
- 'Out of space' when the free list is exhausted (too many digits).
- 'Out of headers' for too many numbers being kept around.
- 'Out of pushdown' for too many items on the stack.
- 'Nesting Depth' for too many levels of nested execution.

NAME

dd - convert and copy a file

SYNOPSIS

dd [option=value] ...

DESCRIPTION

Dd copies the specified input file to the specified output with possible conversions. The standard input and output are used by default. The input and output block size may be specified to take advantage of raw physical I/O.

<i>option</i>	<i>values</i>
if=file	input file name; standard input is default
of=file	output file name; standard output is default
ibs=n	input block size <i>n</i> bytes (default 512)
obs=n	output block size (default 512)
bs=n	set both input and output block size, superseding <i>ibs</i> and <i>obs</i> ; also, if no conversion is specified, it is particularly efficient since no in-core copy need be done
cbs=n	conversion buffer size
skip=n	skip <i>n</i> input records before starting copy
seek=n	seek <i>n</i> records from beginning of output file before copying
count=n	copy only <i>n</i> input records
conv=ascii	convert EBCDIC to ASCII
ebcdic	convert ASCII to EBCDIC
ibm	slightly different map of ASCII to EBCDIC
lcase	map alphabets to lower case
ucase	map alphabets to upper case
swab	swap every pair of bytes
noerror	do not stop processing on an error
sync	pad every input record to <i>ibs</i>
...,...	several comma-separated conversions

Where sizes are specified, a number of bytes is expected. A number may end with **k**, **b**, or **w** to specify multiplication by 1024, 512, or 2 respectively; a pair of numbers may be separated by **x** to indicate a product.

Cbs is used only if *ascii* or *ebcdic* conversion is specified. In the former case *cbs* characters are placed into the conversion buffer, converted to ASCII, and trailing blanks trimmed and new-line added before sending the line to the output. In the latter case ASCII characters are read into the conversion buffer, converted to EBCDIC, and blanks added to make up an output record of size *cbs*.

After completion, *dd* reports the number of whole and partial input and output blocks.

EXAMPLE

This command will read an EBCDIC tape blocked ten 80-byte EBCDIC card images per record into the ASCII file **x**:

```
dd if=/dev/rmt0 of=x ibs=800 cbs=80 conv=ascii,lcase
```

Note the use of raw magtape. *Dd* is especially suited to I/O on the raw physical devices because it allows reading and writing in arbitrary record

sizes.

SEE ALSO

cp(1).

DIAGNOSTICS

f+p records in(out) numbers of full and partial records read(written)

BUGS

The ASCII/EBCDIC conversion tables are taken from the 256 character standard in the CACM Nov, 1968. The *ibm* conversion, while less blessed as a standard, corresponds better to certain IBM print train conventions. There is no universal solution.

New-lines are inserted only on conversion to ASCII; padding is done only on conversion to EBCDIC. These should be separate options.

NAME

delta - make a delta (change) to an SCCS file

SYNOPSIS

delta [-rSID] [-s] [-n] [-glist] [-m[mrlist]] [-y[comment]] [-p] files

DESCRIPTION

Delta is used to permanently introduce into the named SCCS file changes that were made to the file retrieved by *get*(1) (called the *g-file*, or generated file).

Delta makes a delta to each named SCCS file. If a directory is named, *delta* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with *s*.) and unreadable files are silently ignored. If a name of - is given, the standard input is read (see *WARNINGS*); each line of the standard input is taken to be the name of an SCCS file to be processed.

Delta may issue prompts on the standard output depending upon certain keyletters specified and flags (see *admin*(1)) that may be present in the SCCS file (see -m and -y keyletters below).

Keyletter arguments apply independently to each named file.

-rSID

Uniquely identifies which delta is to be made to the SCCS file. The use of this keyletter is necessary only if two or more outstanding *gets* for editing (*get -e*) on the same SCCS file were done by the same person (login name). The SID value specified with the -r keyletter can be either the SID specified on the *get* command line or the SID to be made as reported by the *get* command (see *get*(1)). A diagnostic results if the specified SID is ambiguous, or, if necessary and omitted on the command line.

-s

Suppresses the issue, on the standard output, of the created delta's SID, as well as the number of lines inserted, deleted and unchanged in the SCCS file.

-n

Specifies retention of the edited *g-file* (normally removed at completion of delta processing).

-glist

Specifies a *list* (see *get*(1) for the definition of *list*) of deltas which are to be *ignored* when the file is accessed at the change level (SID) created by this delta.

-m[mrlist]

If the SCCS file has the *v* flag set (see *admin*(1)) then a Modification Request (*MR*) number *must* be supplied as the reason for creating the new delta.

If -m is not used and the standard input is a terminal, the prompt *MRs?* is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. The *MRs?* prompt always precedes the *comments?* prompt (see -y keyletter).

MRs in a list are separated by blanks and/or tab characters. An unescaped new-line character terminates the **MR** list.

Note that if the **v** flag has a value (see *admin(1)*), it is taken to be the name of a program (or shell procedure) which will validate the correctness of the **MR** numbers. If a non-zero exit status is returned from **MR** number validation program, *delta* terminates (it is assumed that the **MR** numbers were not all valid).

-y[comment] Arbitrary text used to describe the reason for making the delta. A null string is considered a valid *comment*.

If **-y** is not specified and the standard input is a terminal, the prompt **comments?** is issued on the standard output before the standard input is read; if the standard input is not a terminal, no prompt is issued. An unescaped new-line character terminates the comment text.

-p Causes *delta* to print (on the standard output) the SCCS file differences before and after the delta is applied in a *diff(1)* format.

FILES

All files of the form *?-file* are explained in the *Source Code Control System User's Guide*. The naming convention for these files is also described there.

<i>g-file</i>	Existed before the execution of <i>delta</i> ; removed after completion of <i>delta</i> .
<i>p-file</i>	Existed before the execution of <i>delta</i> ; may exist after completion of <i>delta</i> .
<i>q-file</i>	Created during the execution of <i>delta</i> ; removed after completion of <i>delta</i> .
<i>x-file</i>	Created during the execution of <i>delta</i> ; renamed to SCCS file after completion of <i>delta</i> .
<i>z-file</i>	Created during the execution of <i>delta</i> ; removed during the execution of <i>delta</i> .
<i>d-file</i>	Created during the execution of <i>delta</i> ; removed after completion of <i>delta</i> .
<i>/usr/bin/bdiff</i>	Program to compute differences between the "gotten" file and the <i>g-file</i> .

WARNINGS

Lines beginning with an SOH ASCII character (binary 001) cannot be placed in the SCCS file unless the SOH is escaped. This character has special meaning to SCCS (see *sccsfile(5)*) and will cause an error.

A *get* of many SCCS files, followed by a *delta* of those files, should be avoided when the *get* generates a large amount of data. Instead, multiple *get/delta* sequences should be used.

If the standard input (**-**) is specified on the *delta* command line, the **-m** (if necessary) and **-y** keyletters *must* also be present. Omission of these keyletters causes an error to occur.

SEE ALSO

admin(1), bdiff(1), get(1), help(1), prs(1), sccsfile(5).

Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

deroff - remove *nroff*/*troff*, *tbl*, and *eqn* constructs

SYNOPSIS

deroff [*-w*] [*-mx*] [*files*]

DESCRIPTION

*Dero**ff* reads each of the *files* in sequence and removes all *troff*(1) requests, macro calls, backslash constructs, *eqn*(1) constructs (between *.EQ* and *.EN* lines, and between delimiters), and *tbl*(1) descriptions, and writes the remainder of the file on the standard output. *Dero**ff* follows chains of included files (*.so* and *.nx troff* commands); if a file has already been included, a *.so* naming that file is ignored and a *.nx* naming that file terminates execution. If no input file is given, *deroff* reads the standard input.

The *-m* option may be followed by an *m*, *s*, or *l*. The resulting *-mm* or *-ms* option causes the *mm* or *ms* macros to be interpreted so that only running text is output (i.e., no text from macro lines.) The *-ml* option forces the *-mm* option and also causes deletion of lists associated with the *mm* macros.

If the *-w* option is given, the output is a word list, one "word" per line, with all other characters deleted. Otherwise, the output follows the original, with the deletions mentioned above. In text, a "word" is any string that *contains* at least two letters and is composed of letters, digits, ampersands (&), and apostrophes ('); in a macro call, however, a "word" is a string that *begins* with at least two letters and contains a total of at least three letters. Delimiters are any characters other than letters, digits, apostrophes, and ampersands. Trailing apostrophes and ampersands are removed from "words."

SEE ALSO

eqn(1), *tbl*(1), *troff*(1).

BUGS

*Dero**ff* is not a complete *troff* interpreter, so it can be confused by subtle constructs. Most such errors result in too much rather than too little output.

The *-ml* option does not handle nested lists correctly.

THE HISTORY OF THE UNITED STATES

OF THE UNITED STATES

The history of the United States is a story of a people who have grown from a small colony of English settlers to a great nation. The story begins with the first English settlers in 1607, who came to America in search of a new home. They found a land of opportunity, but also a land of hardship. The early years were difficult, but the settlers persevered and built a new life for themselves. Over time, the colonies grew and became more independent. They fought for their rights and eventually won their freedom from Britain in 1776. The new nation was born, and it has since grown into a great power.

The United States has a rich and diverse history. It is a land of many cultures, languages, and traditions. The people of the United States have made many contributions to the world, from the arts to science to politics. The history of the United States is a story of a people who have overcome many challenges and built a great nation. The story is still being written, and it is up to us to make sure that it is a story of progress and hope.

THE HISTORY OF THE UNITED STATES

OF THE UNITED STATES

OF THE UNITED STATES

NAME

df - report number of free disk blocks

SYNOPSIS

df [-t] [-f] [file-systems]

DESCRIPTION

Df prints out the number of free blocks and free i-nodes available for on-line file systems by examining the counts kept in the super-blocks; *file-systems* may be specified either by device name (e.g., */dev/rp1*) or by mounted directory name (e.g., */usr*). If the *file-systems* argument is unspecified, the free space on all of the mounted file systems is printed.

The *-t* flag causes the total allocated block figures to be reported as well.

If the *-f* flag is given, only an actual count of the blocks in the free list is made (free i-nodes are not reported). With this option, *df* will report on raw devices.

FILES

*/dev/rf**
*/dev/rk**
*/dev/rp**
/etc/mnttab

SEE ALSO

fsck(8), *fs*(5), *mnttab*(5).

NAME

diction, *explain* - print wordy sentences, interactive thesaurus for *diction*

SYNOPSIS

diction [*-ml*] [*-mm*] [*-n*] [*-f pfile*] file ...
explain

DESCRIPTION

Diction finds all sentences in a document that contain phrases from a data base of bad or wordy *diction*. Each phrase is bracketed with []. Because *diction* runs *deroff* before looking at the text, formatting header files should be included as part of the input. The default macro package *-ms* may be overridden with the flag *-mm*. The flag *-ml* which causes *deroff* to skip lists, should be used if the document contains many lists of non-sentences. The user may supply her/his own pattern file to be used in addition to the default file with *-f pfile*. If the flag *-n* is also supplied the default file will be suppressed.

Explain is an interactive thesaurus for the phrases found by *diction*.

SEE ALSO

deroff(1)

BUGS

Use of non-standard formatting macros may cause incorrect sentence breaks.

1. The first part of the report is a general introduction to the subject.

2. The second part is a detailed description of the methods used.

3. The third part is a discussion of the results obtained, and a comparison with previous work. The fourth part is a conclusion, and a list of references.

4. The fifth part is a summary of the main points of the report.

NAME

diff - differential file comparator

SYNOPSIS

diff [-efbh] file1 file2

DESCRIPTION

Diff tells what lines must be changed in two files to bring them into agreement. If *file1* (*file2*) is '-', the standard input is used. If *file1* (*file2*) is a directory, then a file in that directory whose file-name is the same as the file-name of *file2* (*file1*) is used. The normal output contains lines of these forms:

```
n1 a n3,n4
n1,n2 d n3
n1,n2 c n3,n4
```

These lines resemble *ed* commands to convert *file1* into *file2*. The numbers after the letters pertain to *file2*. In fact, by exchanging 'a' for 'd' and reading backward one may ascertain equally how to convert *file2* into *file1*. As in *ed*, identical pairs where $n1 = n2$ or $n3 = n4$ are abbreviated as a single number.

Following each of these lines come all the lines that are affected in the first file flagged by '<', then all the lines that are affected in the second file flagged by '>'.

The -b option causes trailing blanks (spaces and tabs) to be ignored and other strings of blanks to compare equal.

The -e option produces a script of a, c and d commands for the editor *ed*, which will recreate *file2* from *file1*. The -f option produces a similar script, not useful with *ed*, in the opposite order. In connection with -e, the following shell program may help maintain multiple versions of a file. Only an ancestral file (\$1) and a chain of version-to-version *ed* scripts (\$2,\$3,...) made by *diff* need be on hand. A 'latest version' appears on the standard output.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

Except in rare circumstances, *diff* finds a smallest sufficient set of file differences.

Option -h does a fast, half-hearted job. It works only when changed stretches are short and well separated, but does work on files of unlimited length. Options -e and -f are unavailable with -h.

FILES

```
/tmp/d????
/usr/lib/diffh for -h
```

SEE ALSO

cmp(1), comm(1), ed(1)

DIAGNOSTICS

Exit status is 0 for no differences, 1 for some, 2 for trouble.

BUGS

Editing scripts produced under the -e or -f option are naive about creating lines consisting of a single '.'.

100 100 100

100 100 100

100 100 100

100 100 100

100 100 100

100 100 100

100 100 100

NAME

diff3 - 3-way differential file comparison

SYNOPSIS

diff3 [-ex3] file1 file2 file3

DESCRIPTION

Diff3 compares three versions of a file, and publishes disagreeing ranges of text flagged with these codes:

==== all three files differ

====1 *file1* is different

====2 *file2* is different

====3 *file3* is different

The type of change suffered in converting a given range of a given file to some other is indicated in one of these ways:

f:*n1* a Text is to be appended after line number *n1* in file *f*, where *f* = 1, 2, or 3.

f:*n1*,*n2* c Text is to be changed in the range line *n1* to line *n2*. If *n1* = *n2*, the range may be abbreviated to *n1*.

The original contents of the range follows immediately after a c indication. When the contents of two files are identical, the contents of the lower-numbered file is suppressed.

Under the -e option, *diff3* publishes a script for the editor *ed* that will incorporate into *file1* all changes between *file2* and *file3*, i.e. the changes that normally would be flagged ==== and ====3. Option -x (-3) produces a script to incorporate only changes flagged ==== (====3). The following command will apply the resulting script to 'file1'.

(cat script; echo '1,\$p') | ed - file1

FILES

/tmp/d3?????

/usr/lib/diff3

SEE ALSO

diff(1)

BUGS

Text lines that consist of a single '.' will defeat -e.
Files longer than 64K bytes won't work.

NAME

dircmp - directory comparison

SYNOPSIS

dircmp dir1 dir2

DESCRIPTION

Dircmp examines *dir1* and *dir2* and generates various tabulated information about the contents of the directories. Listings of files that are unique to each directory are generated in addition to a list that indicates whether the files common to both directories have the same contents.

SEE ALSO

cmp(1), diff(1).

NAME

`du` - summarize disk usage

SYNOPSIS

`du [-ars] [names]`

DESCRIPTION

Du gives the number of blocks contained in all files and (recursively) directories within each directory and file specified by the *names* argument. The block count includes the indirect blocks of the file. If *names* is missing, `.` is used.

The optional argument `-s` causes only the grand total (for each of the specified *names*) to be given. The optional argument `-a` causes an entry to be generated for each file. Absence of either causes an entry to be generated for each directory only.

Du is normally silent about directories that cannot be read, files that cannot be opened, etc. The `-r` option will cause *du* to generate messages in such instances.

A file with two or more links is only counted once.

BUGS

If the `-a` option is not used, non-directories given as arguments are not listed.

If there are too many distinct linked files, *du* will count the excess files more than once.

Files with holes in them will get an incorrect block count.

NAME

echo - echo arguments

SYNOPSIS

echo [arg] ...

DESCRIPTION

Echo writes its arguments separated by blanks and terminated by a new-line on the standard output. It also understands C-like escape conventions; beware of conflicts with the shell's use of \:

\b	backspace
\c	print line without new-line
\f	form-feed
\n	new-line
\r	carriage return
\t	tab
\\	backslash
\n	the 8-bit character whose ASCII code is the 1-, 2- or 3-digit octal number <i>n</i> , which must start with a zero.

Echo is useful for producing diagnostics in command files and for sending known data into a pipe.

SEE ALSO

sh(1).

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

NAME

ed, red - text editor

SYNOPSIS

ed [-] [-x] [file]

red [-] [-x] [file]

DESCRIPTION

Ed is the standard text editor. If the *file* argument is given, *ed* simulates an *e* command (see below) on the named file; that is to say, the file is read into *ed*'s buffer so that it can be edited. The optional - suppresses the printing of character counts by *e*, *r*, and *w* commands, of diagnostics from *e* and *q* commands, and of the ! prompt after a *!shell command*. If -x is present, an *x* command is simulated first to handle an encrypted file. *Ed* operates on a copy of the file it is editing; changes made to the copy have no effect on the file until a *w* (write) command is given. The copy of the text being edited resides in a temporary file called the *buffer*. There is only one buffer.

Red is a restricted version of *ed*. It will only allow editing of files in the current directory. It prohibits executing shell commands via *!shell command*. Attempts to bypass these restrictions result in an error message (*restricted shell*).

Both *ed* and *red* support the *fspec*(5) formatting capability. After including a format specification as the first line of *file* and invoking *ed* with your terminal in *stty -tabs* or *stty tab3* mode (see *stty*(1), the specified tab stops will automatically be used when scanning *file*. For example, if the first line of a file contained:

```
<:t5,10,15 s72:>
```

tab stops would be set at columns 5, 10 and 15, and a maximum line length of 72 would be imposed. NOTE: while inputting text, tab characters when typed are expanded to every eighth column as is the default.

Commands to *ed* have a simple and regular structure: zero, one, or two *addresses* followed by a single-character *command*, possibly followed by parameters to that command. These addresses specify one or more lines in the buffer. Every command that requires addresses has default addresses, so that the addresses can very often be omitted.

In general, only one command may appear on a line. Certain commands allow the input of text. This text is placed in the appropriate place in the buffer. While *ed* is accepting text, it is said to be in *input mode*. In this mode, *no* commands are recognized; all input is merely collected. Input mode is left by typing a period (.) alone at the beginning of a line.

Ed supports a limited form of *regular expression* notation; regular expressions are used in addresses to specify lines and in some commands (e.g., *s*) to specify portions of a line that are to be substituted. A regular expression (RE) specifies a set of character strings. A member of this set of strings is said to be *matched* by the RE. The REs allowed by *ed* are constructed as follows:

The following *one-character REs* match a *single* character:

- 1.1 An ordinary character (*not* one of those discussed in 1.2 below) is a one-character RE that matches itself.

- 1.2 A backslash (\) followed by any special character is a one-character RE that matches the special character itself. The special characters are:
- a. ., *, [, and \ (period, asterisk, left square bracket, and backslash, respectively), which are always special, *except* when they appear within square brackets ([]); see 1.4 below).
 - b. ^ (caret or circumflex), which is special at the *beginning* of an *entire* RE (see 3.1 and 3.2 below), or when it immediately follows the left of a pair of square brackets ([]) (see 1.4 below).
 - c. \$ (currency symbol), which is special at the *end* of an entire RE (see 3.2 below).
 - d. The character used to bound (i.e., delimit) an entire RE, which is special for that RE (for example, see how slash (/) is used in the *g* command, below.)
- 1.3 A period (.) is a one-character RE that matches any character except new-line.
- 1.4 A non-empty string of characters enclosed in square brackets ([]) is a one-character RE that matches *any one* character in that string. If, however, the first character of the string is a circumflex (^), the one-character RE matches any character *except* new-line and the remaining characters in the string. The ^ has this special meaning *only* if it occurs first in the string. The minus (-) may be used to indicate a range of consecutive ASCII characters; for example, [0-9] is equivalent to [0123456789]. The - loses this special meaning if it occurs first (after an initial ^, if any) or last in the string. The right square bracket (]) does not terminate such a string when it is the first character within it (after an initial ^, if any); e.g., []a-f] matches either a right square bracket (]) or one of the letters a through f inclusive. The four characters listed in 1.2.a above stand for themselves within such a string of characters.

The following rules may be used to construct REs from one-character REs:

- 2.1 A one-character RE is a RE that matches whatever the one-character RE matches.
- 2.2 A one-character RE followed by an asterisk (*) is a RE that matches *zero* or more occurrences of the one-character RE. If there is any choice, the longest leftmost string that permits a match is chosen.
- 2.3 A one-character RE followed by \{m\}, \{m,\}, or \{m,n\} is a RE that matches a *range* of occurrences of the one-character RE. The values of *m* and *n* must be non-negative integers less than 256; \{m\} matches *exactly* *m* occurrences; \{m,\} matches *at least* *m* occurrences; \{m,n\} matches *any number* of occurrences *between* *m* and *n* inclusive. Whenever a choice exists, the RE matches as many occurrences as possible.
- 2.4 The concatenation of REs is a RE that matches the concatenation of the strings matched by each component of the RE.

- 2.5 A RE enclosed between the character sequences `\(` and `\)` is a RE that matches whatever the unadorned RE matches.
- 2.6 The expression `\n` matches the same string of characters as was matched by an expression enclosed between `\(` and `\)` *earlier* in the same RE. Here *n* is a digit; the sub-expression specified is that beginning with the *n*-th occurrence of `\(` (counting from the left). For example, the expression `^\.*\)\1$` matches a line consisting of two repeated appearances of the same string.

Finally, an *entire RE* may be constrained to match only an initial segment or final segment of a line (or both):

- 3.1 A circumflex (`^`) at the beginning of an entire RE constrains that RE to match an *initial* segment of a line.
- 3.2 A currency symbol (`$`) at the end of an entire RE constrains that RE to match a *final* segment of a line.

The construction `^entire RE$` constrains the entire RE to match the entire line.

The null RE (e.g., `/`) is equivalent to the last RE encountered. See also the last paragraph before *FILES* below.

To understand addressing in *ed* it is necessary to know that at any time there is a *current line*. Generally speaking, the current line is the last line affected by a command; the exact effect on the current line is discussed under the description of each command. *Addresses* are constructed as follows:

1. The character `.` addresses the current line.
2. The character `$` addresses the last line of the buffer.
3. A decimal number *n* addresses the *n*-th line of the buffer.
4. `'x` addresses the line marked with the mark name character *x*, which must be a lower-case letter. Lines are marked with the *k* command described below.
5. A RE enclosed by slashes (`/`) addresses the first line found by searching *forward* from the line *following* the current line toward the end of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the beginning of the buffer and continues up to and including the current line, so that the entire buffer is searched. See also the last paragraph before *FILES* below.
6. A RE enclosed in question marks (`?`) addresses the first line found by searching *backward* from the line *preceding* the current line toward the beginning of the buffer and stopping at the first line containing a string matching the RE. If necessary, the search wraps around to the end of the buffer and continues up to and including the current line. See also the last paragraph before *FILES* below.
7. An address followed by a plus sign (`+`) or a minus sign (`-`) followed by a decimal number specifies that address plus (respectively minus) the indicated number of lines. The plus sign may be omitted.

8. If an address begins with + or -, the addition or subtraction is taken with respect to the current line; e.g., -5 is understood to mean .-5.
9. If an address ends with + or -, then 1 is added to or subtracted from the address, respectively. As a consequence of this rule and of rule 8 immediately above, the address - refers to the line preceding the current line. (To maintain compatibility with earlier versions of the editor, the character ^ in addresses is entirely equivalent to -.) Moreover, trailing + and - characters have a cumulative effect, so -- refers to the current line less 2.
10. For convenience, a comma (,) stands for the address pair 1,\$, while a semicolon (;) stands for the pair .,\$.

Commands may require zero, one, or two addresses. Commands that require no addresses regard the presence of an address as an error. Commands that accept one or two addresses assume default addresses when an insufficient number of addresses is given; if more addresses are given than such a command requires, the last one(s) are used.

Typically, addresses are separated from each other by a comma (,). They may also be separated by a semicolon (;). In the latter case, the current line (.) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules 5. and 6. above). The second address of any two-address sequence must correspond to a line that follows, in the buffer, the line corresponding to the first address.

In the following list of *ed* commands, the default addresses are shown in parentheses. The parentheses are *not* part of the address; they show that the given addresses are the default.

It is generally illegal for more than one command to appear on a line. However, any command (except *e*, *f*, *r*, or *w*) may be suffixed by *l*, *n* or *p*, in which case the current line is either listed, numbered or printed, respectively, as discussed below under the *l*, *n* and *p* commands.

(.)a
<text>

The *a*ppend command reads the given text and appends it after the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. Address 0 is legal for this command: it causes the "appended" text to be placed at the beginning of the buffer. The maximum number of characters that may be entered from a terminal is 256 per line (including the newline character).

(.)c
<text>

The *c*hange command deletes the addressed lines, then accepts input text that replaces these lines; . is left at the last line input, or, if there were none, at the first line that was not deleted.

(...)d

The *d*elete command deletes the addressed lines from the buffer.

The line after the last line deleted becomes the current line; if the lines deleted were originally at the end of the buffer, the new last line becomes the current line.

e file

The *e* edit command causes the entire contents of the buffer to be deleted, and then the named file to be read in; *.* is set to the last line of the buffer. If no file name is given, the currently-remembered file name, if any, is used (see the *f* command). The number of characters read is typed; *file* is remembered for possible use as a default file name in subsequent *e*, *r*, and *w* commands. If *file* is replaced by *!*, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. Such a shell command is *not* remembered as the current file name. See also *DIAGNOSTICS* below.

E file

The *Edit* command is like *e*, except that the editor does not check to see if any changes have been made to the buffer since the last *w* command.

f file

If *file* is given, the *f* file-name command changes the currently-remembered file name to *file*; otherwise, it prints the currently-remembered file name.

(1,3)g/RE/command list

In the global command, the first step is to mark every line that matches the given RE. Then, for every such line, the given *command list* is executed with *.* initially set to that line. A single command or the first of a list of commands appears on the same line as the global command. All lines of a multi-line list except the last line must be ended with a **; *a*, *i*, and *c* commands and associated input are permitted; the *.* terminating input mode may be omitted if it would be the last line of the *command list*. An empty *command list* is equivalent to the *p* command. The *g*, *G*, *v*, and *V* commands are *not* permitted in the *command list*. See also *BUGS* and the last paragraph before *FILES* below.

(1,3)G/RE/

In the interactive Global command, the first step is to mark every line that matches the given RE. Then, for every such line, that line is printed, *.* is changed to that line, and any *one* command (other than one of the *a*, *c*, *i*, *g*, *G*, *v*, and *V* commands) may be input and is executed. After the execution of that command, the next marked line is printed, and so on; a new-line acts as a null command; an *&* causes the re-execution of the most recent command executed within the current invocation of *G*. Note that the commands input as part of the execution of the *G* command may address and affect *any* lines in the buffer. The *G* command can be terminated by an interrupt signal (ASCII DEL or BREAK).

h

The *help* command gives a short error message that explains the reason for the most recent ? diagnostic.

H

The *Help* command causes *ed* to enter a mode in which error messages are printed for all subsequent ? diagnostics. It will also explain the previous ? if there was one. The *H* command alternately turns this mode on and off; it is initially off.

(.)i
<text>

The *insert* command inserts the given text before the addressed line; . is left at the last inserted line, or, if there were none, at the addressed line. This command differs from the *a* command only in the placement of the input text. Address 0 is not legal for this command. The maximum number of characters that may be entered from a terminal is 256 per line (including the newline character).

(...+1)j

The *join* command joins contiguous lines by removing the appropriate new-line characters. If exactly one address is given, this command does nothing.

(.)kx

The *mark* command marks the addressed line with name *x*, which must be a lower-case letter. The address '*x*' then addresses this line; . is unchanged.

(...)l

The *list* command prints the addressed lines in an unambiguous way: a few non-printing characters (e.g., *tab*, *backspace*) are represented by (hopefully) mnemonic overstrikes, all other non-printing characters are printed in octal, and long lines are folded. An *l* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(...)ma

The *move* command repositions the addressed line(s) after the line addressed by *a*. Address 0 is legal for *a* and causes the addressed line(s) to be moved to the beginning of the file; it is an error if address *a* falls within the range of moved lines; . is left at the last line moved.

(...)n

The *number* command prints the addressed lines, preceding each line by its line number and a tab character; . is left at the last line printed. The *n* command may be appended to any other command other than *e*, *f*, *r*, or *w*.

(...)p

The *print* command prints the addressed lines; . is left at the last line printed. The *p* command may be appended to any other command other than *e*, *f*, *r*, or *w*; for example, *dp* deletes the current line and prints the new current line.

P

The editor will prompt with a * for all subsequent commands. The *P* command alternately turns this mode on and off; it is initially

off.

q

The quit command causes *ed* to exit. No automatic write of a file is done (but see *DIAGNOSTICS* below).

Q

The editor exits without checking if changes have been made in the buffer since the last *w* command.

(**s**)*r file*

The read command reads in the given file after the addressed line. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands). The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. Address 0 is legal for *r* and causes the file to be read at the beginning of the buffer. If the read is successful, the number of characters read is typed; . is set to the last line read in. If *file* is replaced by !, the rest of the line is taken to be a shell (*sh*(1)) command whose output is to be read. For example, "*\$r !ls*" appends current directory to the end of the file being edited. Such a shell command is *not* remembered as the current file name.

(...)*s/RE/replacement /* or
(...)*s/RE/replacement /g*

The substitute command searches each addressed line for an occurrence of the specified RE. In each line in which a match is found, all (non-overlapped) matched strings are replaced by the *replacement* if the global replacement indicator *g* appears after the command. If the global indicator does not appear, only the first occurrence of the matched string is replaced. It is an error for the substitution to fail on *all* addressed lines. Any character other than space or new-line may be used instead of / to delimit the RE and the *replacement*; . is left at the last line on which a substitution occurred. See also the last paragraph before *FILES* below.

An ampersand (&) appearing in the *replacement* is replaced by the string matching the RE on the current line. The special meaning of & in this context may be suppressed by preceding it by \. As a more general feature, the characters \n, where *n* is a digit, are replaced by the text matched by the *n*-th regular subexpression of the specified RE enclosed between \ (and \). When nested parenthesized subexpressions are present, *n* is determined by counting occurrences of \ (starting from the left. When the character % is the only character in the *replacement*, the *replacement* used in the most recent substitute command is used as the *replacement* in the current substitute command. The % loses its special meaning when it is in a replacement string of more than one character or is preceded by a \.

A line may be split by substituting a new-line character into it. The new-line in the *replacement* must be escaped by preceding it by \. Such substitution cannot be done as part of a *g* or *v*

command list.

(...)*ta*

This command acts just like the *m* command, except that a *copy* of the addressed lines is placed after address *a* (which may be 0); . is left at the last line of the copy.

u

The *undo* command nullifies the effect of the most recent command that modified anything in the buffer, namely the most recent *a*, *c*, *d*, *g*, *i*, *j*, *m*, *r*, *s*, *t*, *v*, *G*, or *V* command.

(1,*s*)*v*/*RE*/*command list*

This command is the same as the global command *g* except that the *command list* is executed with . initially set to every line that does *not* match the *RE*.

(1,*s*)*V*/*RE*/

This command is the same as the interactive global command *G* except that the lines that are marked during the first step are those that do *not* match the *RE*.

(1,*s*)*w* *file*

The *write* command writes the addressed lines into the named *file*. If the file does not exist, it is created with mode 666 (readable and writable by everyone), unless your *umask* setting (see *sh*(1)) dictates otherwise. The currently-remembered file name is *not* changed unless *file* is the very first file name mentioned since *ed* was invoked. If no file name is given, the currently-remembered file name, if any, is used (see *e* and *f* commands); . is unchanged. If the command is successful, the number of characters written is typed. If *file* is replaced by *!*, the rest of the line is taken to be a shell (*sh*(1)) command whose standard input is the addressed lines. Such a shell command is *not* remembered as the current file name.

X

A key string is demanded from the standard input. Subsequent *e*, *r*, and *w* commands will encrypt and decrypt the text with this key by the algorithm of *crypt*(1). An explicitly empty key turns off encryption.

(*s*)=

The line number of the addressed line is typed; . is unchanged by this command.

!shell command

The remainder of the line after the *!* is sent to the UNIX System shell (*sh*(1)) to be interpreted as a command. Within the text of that command, the unescaped character *%* is replaced with the remembered file name; if a *!* appears as the first character of the shell command, it is replaced with the text of the previous shell command. Thus, *!!* will repeat the last shell command. If any expansion is performed, the expanded line is echoed; . is unchanged.

(.+1)<new-line>

An address alone on a line causes the addressed line to be printed. A new-line alone is equivalent to `.+1p`; it is useful for stepping forward through the buffer.

If an interrupt signal (ASCII DEL or BREAK) is sent, `ed` prints a `?` and returns to *its* command level.

Some size limitations: 512 characters per line, 256 characters per global command list, 64 characters per file name, and 128K characters in the buffer. The limit on the number of lines depends on the amount of user memory: each line takes 1 word.

When reading a file, `ed` discards ASCII NUL characters and all characters after the last new-line. Files (e.g., `a.out`) that contain characters not in the ASCII set (bit 8 on) cannot be edited by `ed`.

If the closing delimiter of a RE or of a replacement string (e.g., `/`) would be the last character before a new-line, that delimiter may be omitted, in which case the addressed line is printed. The following pairs of commands are equivalent:

`s/s1/s2` `s/s1/s2/p`

`g/s1` `g/s1/p`

`?s1` `?s1?`

FILES

`/tmp/e#`

temporary; `#` is the process number.

`ed.hup` work is saved here if the terminal is hung up.

DIAGNOSTICS

`?` for command errors.

`?file` for an inaccessible file.

(use the `help` and `Help` commands for detailed explanations).

If changes have been made in the buffer since the last `w` command that wrote the entire buffer, `ed` warns the user if an attempt is made to destroy `ed`'s buffer via the `e` or `q` commands: it prints `?` and allows one to continue editing. A second `e` or `q` command at this point will take effect. The `-` command-line option inhibits this feature.

SEE ALSO

`crypt(1)`, `grep(1)`, `sed(1)`, `sh(1)`, `stty(1)`, `fspec(5)`, `regex(7)`.

A Tutorial Introduction to the UNIX Text Editor by B. W. Kernighan.

Advanced Editing on UNIX by B. W. Kernighan.

CAVEATS AND BUGS

A `!` command cannot be subject to a `g` or a `v` command.

The `!` command and the `!` escape from the `e`, `r`, and `w` commands cannot be used if the editor is invoked from a restricted shell (see `sh(1)`).

The sequence `\n` in a RE does not match a new-line character.

The `l` command mishandles DEL.

Files encrypted directly with the `crypt(1)` command with the null key cannot be edited.

Characters are masked to 7 bits on input.

NAME

efl - Extended Fortran Language

SYNOPSIS

efl [options] [files]

DESCRIPTION

Efl compiles a program written in the EFL language into clean Fortran on the standard output. *Efl* provides the C-like control constructs of *rat-for*(1):

statement grouping with braces.

decision-making:

if, **if-else**, and **select-case** (also known as **switch-case**);
while, **for**, Fortran **do**, **repeat**, and **repeat ... until** loops;
 multi-level **break** and **next**.

EFL has C-like data structures, e.g.:

```
struct
{
  integer flags(3)
  character(8) name
  long real coords(2)
  } table(100)
```

The language offers generic functions, assignment operators (**+=**, **&=**, etc.), and sequentially evaluated logical operators (**&&** and **||**). There is a uniform input/output syntax:

```
write(6,x,y:f(7,2), do i=1,10 { a(i,j),z.b(i) })
```

EFL also provides some syntactic "sugar":

free-form input:

multiple statements per line; automatic continuation; statement label names (not just numbers).

comments:

this is a comment.

translation of relational and logical operators:

>, **>=**, **&**, etc., become **.GT.**, **.GE.**, **.AND.**, etc.

return expression to caller from function:

return (*expression*)

defines:

define *name replacement*

includes:

include *file*

Efl understands several option arguments: **-w** suppresses warning messages, **-#** suppresses comments in the generated program, and the default option **-C** causes comments to be included in the generated program.

An argument with an embedded **=** (equal sign) sets an EFL option as if it had appeared in an **option** statement at the start of the program. Many options are described in the reference manual. A set of defaults for a

particular target machine may be selected by one of the choices: **system=unix**, **system=gc**, or **system=cray**. The default setting of the **system** option is the same as the machine the compiler is running on. Other specific options determine the style of input/output, error handling, continuation conventions, the number of characters packed per word, and default formats.

Efl is best used with *f77(1)*.

SEE ALSO

cc(1), *f77(1)*, *ratfor(1)*.

The Programming Language EFL by S.I. Feldman.

NAME

emacs - a screen editor

SYNOPSIS

emacs files...

DESCRIPTION

Unix *Emacs* is a text editor styled after the editors of the same name that exist on ITS, Twenex and Multics. Invoking *emacs* with a list of files causes *emacs* to start up and do a visit-file on each of the named files. For more information, read the manual.

FILES

/usr/lib/emacs/*

ENVIRONMENT

EPATH is a path used when locating files with the load command.

SEE ALSO

The Unix Emacs manual, which should be on /usr/doc/emacs

BUGS

Probably many on the CADMUS. Emacs needs virtual memory, as it grows very big.

HISTORY

08-Jan-81 James Gosling (jag) at Carnegie-Mellon University
Created.

The first of these is the fact that the
the second is the fact that the
the third is the fact that the

the fourth is the fact that the

the fifth is the fact that the

the sixth is the fact that the

the seventh is the fact that the

NAME

env - set environment for command execution

SYNOPSIS

env [-] [name=value] ... [command args]

DESCRIPTION

Env obtains the current *environment*, modifies it according to its arguments, then executes the command with the modified environment. Arguments of the form *name=value* are merged into the inherited environment before the command is executed. The - flag causes the inherited environment to be ignored completely, so that the command is executed with exactly the environment specified by the arguments.

If no command is specified, the resulting environment is printed, one name-value pair per line.

SEE ALSO

sh(1), exec(2), profile(5), environ(7).

NAME

eqn, neqn, checkeq - format mathematical text for nroff or troff

SYNOPSIS

```
eqn [ -dxy ] [ -pn ] [ -sn ] [ -fn ] [ files ]
neqn [ -dxy ] [ -pn ] [ -sn ] [ -fn ] [ files ]
checkeq [ files ]
```

DESCRIPTION

Eqn is a *troff*(1) preprocessor for typesetting mathematical text on a Wang Laboratories, Inc. C/A/T phototypesetter, while *neqn* is used for the same purpose with *nroff*(1) on typewriter-like terminals. Usage is almost always:

```
eqn files | troff
neqn files | nroff
```

or equivalent.

If no files are specified, these programs read from the standard input. A line beginning with *.EQ* marks the start of an equation; the end of an equation is marked by a line beginning with *.EN*. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to designate two characters as *delimiters*; subsequent text between delimiters is then treated as *eqn* input. Delimiters may be set to characters *x* and *y* with the command-line argument *-dxy* or (more commonly) with *delim xy* between *.EQ* and *.EN*. The left and right delimiters may be the same character; the dollar sign is often used as such a delimiter. Delimiters are turned off by *delim off*. All text that is neither between delimiters nor between *.EQ* and *.EN* is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and *.EQ/.EN* pairs.

Tokens within *eqn* are separated by spaces, tabs, new-lines, braces, double quotes, tildes, and circumflexes. Braces {} are used for grouping; generally speaking, anywhere a single character such as *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde (~) represents a full space in the output, circumflex (^) half as much.

Subscripts and superscripts are produced with the keywords *sub* and *sup*. Thus *x sub j* makes x_j , *a sub k sup 2* produces a_k^2 , while $e^{x^2+y^2}$ is made with *e sup {x sup 2 + y sup 2}*. Fractions are made with *over*: *a over b* yields $\frac{a}{b}$; *sqrt* makes square roots: *1 over sqrt {ax sup 2+bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords *from* and *to* introduce lower and upper limits: $\lim_{n \rightarrow \infty} \sum_0^n x_i$ is made with *lim from {n -> inf} sum from 0 to n x sub i*. Left and right brackets, braces, etc., of the right height are made with *left* and *right*: *left [x sup 2 + y sup 2 over alpha right] ~ = ~ 1* produces $\left[x^2 + \frac{y^2}{\alpha} \right] = 1$. Legal characters after *left* and *right* are braces, brackets, bars, c and f

for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket). A *left thing* need not have a matching *right thing*.

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: **pile** {*a above b above c*} produces $\begin{smallmatrix} a \\ b \\ c \end{smallmatrix}$. Piles may have arbitrary numbers of elements; **lpile** left-justifies, **pile** and **cpile** center (but with different vertical spacing), and **rpile** right justifies. Matrices are made with **matrix**:

matrix {*lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{smallmatrix} x_i & 1 \\ y_2 & 2 \end{smallmatrix}$.

In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t)* bar is $\bar{x} = \overline{f(t)}$, *y dotdot bar ~ = ~ n under* is $\bar{y} = \underline{n}$, and *x vec ~ = ~ y dyad* is $\vec{x} = \vec{y}$.

Point sizes and fonts can be changed with **size** *n* or **size** $\pm n$, **roman**, **italic**, **bold**, and **font** *n*. Point sizes and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **-sn** and **-fn**.

Normally, subscripts and superscripts are reduced by 3 points from the previous size; this may be changed by the command-line argument **-pn**.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**:

define *thing* % replacement %

defines a new token called *thing* that will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords such as **sum** (Σ), **int** (\int), **inf** (∞), and shorthands such as **>=** (\geq), **!=** (\neq), and **->** (\rightarrow) are recognized. Greek letters are spelled out in the desired case, as in **alpha** (α), or **GAMMA** (Γ). Mathematical words such as **sin**, **cos**, and **log** are made Roman automatically. **Troff**(1) four-character escapes such as **\(dd** (\ddagger) and **\(bs** (\textcircled{b}) may be used anywhere. Strings enclosed in double quotes ("...") are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with **troff**(1) when all else fails. Full details are given in the manual cited below.

SEE ALSO

Typesetting Mathematics—User's Guide by B. W. Kernighan and L. L. Cherry.

New Graphic Symbols for EQN and NEQN by C. Scrocca.

mm(1), **mmt**(1), **tbl**(1), **troff** (1), **eqnchar**(7), **mm**(7), **mv**(7).

BUGS

To embolden digits, parentheses, etc., it is necessary to quote them, as in **bold** "12.3".

See also **BUGS** under **troff**(1).

NAME

error - analyze and disperse compiler error messages

SYNOPSIS

```
error [ -n ] [ -s ] [ -q ] [ -v ] [ -t suffixlist ] [ -I ignorefile ] [
name ]
```

DESCRIPTION

Error analyzes and optionally disperses the diagnostic error messages produced by a number of compilers and language processors to the source file and line where the errors occurred. It can replace the painful, traditional methods of scribbling abbreviations of errors on paper, and permits error messages and source code to be viewed simultaneously without machinations of multiple windows in a screen editor.

Error looks at the error messages, either from the specified file *name* or from the standard input, and attempts to determine which language processor produced each error message, determines the source file and line number to which the error message refers, determines if the error message is to be ignored or not, and inserts the (possibly slightly modified) error message into the source file as a comment on the line preceeding to which the line the error message refers. Error messages which can't be categorized by language processor or content are not inserted into any file, but are sent to the standard output. *Error* touches source files only after all input has been read. By specifying the *-q* query option, the user is asked to confirm any potentially dangerous (such as touching a file) or verbose action. Otherwise *error* proceeds on its merry business. If the *-t* touch option and associated suffix list is given, *error* will restrict itself to touch only those files with suffices in the suffix list. Error also can be asked (by specifying *-v*) to invoke *vi*(1) on the files in which error messages were inserted; this obviates the need to remember the names of the files with errors.

Error is intended to be run with its standard input connected via a pipe to the error message source. Some language processors put error messages on their standard error file; others put their messages on the standard output. Hence, both error sources should be piped together into *error*. For example, when using the *cs*h syntax,

```
make |& error -q -v
```

or, for the Bourne shell,

```
make 2>&1 | error -q -v
```

will analyze all the error messages produced by whatever programs *make* runs.

Error knows about the error messages produced by: *make*, *cc*, *cpp*, *ccom*, *as*, *ld*, *lint*, *pi*, *pc* and *f77*. *Error* knows a standard format for error messages produced by the language processors, so is sensitive to changes in these formats. For all languages except *Pascal*, error messages are restricted to be on one line. Some error messages refer to more than one line in more than one files; *error* will duplicate the error message and insert it at all of the places referenced.

Error will do one of six things with error messages.

synchronize

Some language processors produce short errors describing which file it is processing. *Error* uses these to determine the file name for languages that don't include the file name in each error message. These synchronization messages are consumed entirely by *error*.

discard Error messages from *lint* that refer to one of the two *lint* libraries, */usr/lib/lldlib-lc* and */usr/lib/lldlib-port* are discarded, to prevent accidentally touching these libraries. Again, these error messages are consumed entirely by *error*.

nullify Error messages from *lint* can be nullified if they refer to a specific function, which is known to generate diagnostics which are not interesting. Nullified error messages are not inserted into the source file, but are written to the standard output. The names of functions to ignore are taken from either the file named *.errorrc* in the user's home directory, or from the file named by the *-I* option. If the file does not exist, no error messages are nullified. If the file does exist, there must be one function name per line.

not file specific

Error messages that can't be intuited are grouped together, and written to the standard output before any files are touched. They will not be inserted into any source file.

file specific

Error messages that refer to a specific file, but to no specific line, are written to the standard output when that file is touched.

true errors

Error messages that can be intuited are candidates for insertion into the file to which they refer.

Only true error messages are candidates for inserting into the file they refer to. Other error messages are consumed entirely by *error* or are written to the standard output. *Error* inserts the error messages into the source file on the line preceeding the line the language processor found in error. Each error message is turned into a one line comment for the language, and is internally flagged with the string "###" at the beginning of the error, and "%%" at the end of the error. This makes pattern searching for errors easier with an editor, and allows the messages to be easily removed. In addition, each error message contains the source line number for the line the message refers to. A reasonably formatted source program can be recompiled with the error messages still in it, without having the error messages themselves cause future errors. For poorly formatted source programs in free format languages, such as C or Pascal, it is possible to insert a comment into another comment, which can wreak havoc with a future compilation. To avoid this, format the source program so there are no language statements on the same line as the end of a comment.

Options available with *error* are:

- n Do *not* touch any files; all error messages are sent to the standard output.
- q The user is *queried* whether s/he wants to touch the file. A "y" or "n" to the question is necessary to continue. Absence of the -q option implies that all referenced files (except those referring to discarded error messages) are to be touched.
- v After all files have been touched, overlay the visual editor *vi* with it set up to edit all files touched, and positioned in the first touched file at the first error. If *vi* can't be found, try *ex* or *ed* from standard places.
- t Take the following argument as a suffix list. Files whose suffices do not appear in the suffix list are not touched. The suffix list is dot seperated, and "*" wildcards work. Thus the suffix list:

".c.y.foo*.h"

allows *error* to touch files ending with ".c", ".y", ".foo*" and ".y".

- s Print out *statistics* regarding the error categorization. Not too useful.

Error catches interrupt and terminate signals, and if in the insertion phase, will orderly terminate what it is doing.

AUTHOR

Robert Henry

FILES

~/errorrc	function names to ignore for <i>lint</i> error messages
/dev/tty	user's teletype

BUGS

Opens the teletype directly to do user querying.

Source files with links make a new copy of the file with only one link to it.

Changing a language processor's format of error messages may cause *error* to not understand the error message.

Error, since it is purely mechanical, will not filter out subsequent errors caused by 'floodgating' initiated by one syntactically trivial error. Humans are still much better at discarding these related errors.

Pascal error messages belong after the lines affected (error puts them before). The alignment of the '|' marking the point of error is also disturbed by *error*.

Error was designed for work on CRT's at reasonably high speed. It is less pleasant on slow speed terminals, and has never been used on hardcopy terminals.

The first of these is the fact that the...

The second of these is the fact that the...

The third of these is the fact that the...

The fourth of these is the fact that the...

The fifth of these is the fact that the...

The sixth of these is the fact that the...

The seventh of these is the fact that the...

The eighth of these is the fact that the...

The ninth of these is the fact that the...

The tenth of these is the fact that the...

The eleventh of these is the fact that the...

The twelfth of these is the fact that the...

The thirteenth of these is the fact that the...

The fourteenth of these is the fact that the...

NAME

`ex` - text editor

SYNOPSIS

`ex [-] [-v] [-t tag] [-r] [+command] [-l] [-x] name ...`

DESCRIPTION

Ex is the root of a family of editors: *ex* and *vi*. *Ex* is a superset of *ed*, with the most notable extension being a display editing facility. Display based editing is the focus of *vi*.

If you have a CRT terminal, you may wish to use a display based editor; in this case see *vi*(1), which is a command which focuses on the display editing portion of *ex*.

DOCUMENTATION

The *Ex Reference Manual* is a comprehensive and complete manual for the command mode features of *ex*, but you cannot learn to use the editor by reading it. For an introduction to more advanced forms of editing using the command mode of *ex* see the editing documents written by Brian Kernighan for the editor *ed*; the material in the introductory and advanced documents works also with *ex*.

An Introduction to Display Editing with Vi introduces the display editor *vi* and provides reference material on *vi*. The *Vi Quick Reference* card summarizes the commands of *vi* in a useful, functional way, and is useful with the *Introduction*.

FOR ED USERS

If you have used *ed* you will find that *ex* has a number of new features useful on CRT terminals. Intelligent terminals and high speed terminals are very pleasant to use with *vi*. Generally, the editor uses far more of the capabilities of terminals than *ed* does, and uses the terminal capability data base *termcap*(5) and the type of the terminal you are using from the variable *TERM* in the environment to determine how to drive your terminal efficiently. The editor makes use of features such as insert and delete character and line in its **visual** command (which can be abbreviated **vi**) and which is the central mode of editing when using *vi*(1). There is also an interline editing **open** (**o**) command which works on all terminals.

Ex contains a number of new features for easily viewing the text of the file. The **z** command gives easy access to windows of text. Hitting **^D** causes the editor to scroll a half-window of text and is more useful for quickly stepping through a file than just hitting return. Of course, the screen oriented **visual** mode gives constant access to editing context.

Ex gives you more help when you make mistakes. The **undo** (**u**) command allows you to reverse any single change which goes astray. *Ex* gives you a lot of feedback, normally printing changed lines, and indicates when more than a few lines are affected by a command so that it is easy to detect when a command has affected more lines than it should have.

The editor also normally prevents overwriting existing files unless you edited them so that you don't accidentally clobber with a *write* a file other than the one you are editing. If the system (or editor) crashes, or you accidentally hang up the phone, you can use the editor **recover**

command to retrieve your work. This will get you back to within a few lines of where you left off.

Ex has several features for dealing with more than one file at a time. You can give it a list of files on the command line and use the **next (n)** command to deal with each in turn. The **next** command can also be given a list of file names, or a pattern as used by the shell to specify a new set of files to be dealt with. In general, filenames in the editor may be formed with full shell metasyntax. The metacharacter '%' is also available in forming filenames and is replaced by the name of the current file.

For moving text between files and within a file the editor has a group of buffers, named *a* through *z*. You can place text in these named buffers and carry it over when you edit another file.

There is a command **&** in *ex* which repeats the last **substitute** command. In addition there is a confirmed substitute command. You give a range of substitutions to be done and the editor interactively asks whether each substitution is desired.

It is possible to ignore case of letters in searches and substitutions. *Ex* also allows regular expressions which match words to be constructed. This is convenient, for example, in searching for the word "edit" if your document also contains the word "editor."

Ex has a set of *options* which you can set to tailor it to your liking. One option which is very useful is the *autoindent* option which allows the editor to automatically supply leading white space to align text. You can then use the ^D key as a backtab and space and tab forward to align new code easily.

Miscellaneous new useful features include an intelligent **join (j)** command which supplies white space between joined lines automatically, commands **<** and **>** which shift groups of lines, and the ability to filter portions of the buffer through commands such as *sort*.

INVOCATION OPTIONS

The following invocation options are interpreted by *ex*:

- Suppress all interactive-user feedback. This is useful in processing editor scripts.
- v Invokes *vi*
- ttag Edit the file containing the *tag* and position the editor at its definition.
- rfile Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
- +command Begin editing by executing the specified editor search or positioning *command*.
- l LISP mode; indents appropriately for lisp code, the **() {} []** and **]** commands in *vi* and *open* are modified to have meaning for *lisp*.
- x Encryption mode; a key is prompted for allowing creation or editing of an encrypted file.

The *name* argument indicates files to be edited.

Ex States

Command Normal and initial state. Input prompted for by `:`. Your kill character cancels partial command.

Insert Entered by `a i` and `c`. Arbitrary text may be entered. Insert is normally terminated by line having only `.` on it, or abnormally with an interrupt.

Open/visual Entered by `open` or `vi`, terminates with `Q` or `^\`.

Ex command names and abbreviations

abbrev	ab	next	n	unabbrev	una
append	a	number	nu	undo	u
args	ar	open	o	unmap	unm
change	c	preserve	pre	version	ve
copy	co	print	p	visual	vi
delete	d	put	pu	write	w
edit	e	quit	q	xit	x
file	f	read	re	yank	ya
global	g	recover	rec	window	z
insert	i	rewind	rew	escape	!
join	j	set	se	lshift	<
list	l	shell	sh	print next	CR
map		source	so	resubst	&
mark	ma	stop	st	rshift	>
move	m	substitute	s	scroll	~D

Ex Command Addresses

n	line <i>n</i>	/pat	next with <i>pat</i>
.	current	?pat	previous with <i>pat</i>
\$	last	x-n	<i>n</i> before <i>x</i>
+	next	x,y	<i>x</i> through <i>y</i>
-	previous	'x	marked with <i>x</i>
+n	<i>n</i> forward	"	previous context
%	1,\$		

Initializing options

EXINIT	place set's here in environment var.
\$HOME/.exrc	editor initialization file
./exrc	editor initialization file
set x	enable option
set nox	disable option
set x=val	give value <i>val</i>
set	show changed options
set all	show all options
set x?	show value of option <i>x</i>

Useful options

autoindent	ai	supply indent
autowrite	aw	write before changing files
ignorecase	ic	in scanning
lisp		(){} are s-exp's
list		print ~l for tab, \$ at end
magic		. [* special in patterns

number	nu	number lines
paragraphs	para	macro names which start ...
redraw		simulate smart terminal
scroll		command mode lines
sections	sect	macro names ...
shiftwidth	sw	for < >, and input ~D
showmatch	sm	to) and } as typed
slowopen	slow	stop updates during insert
window		visual mode lines
wrapsan	ws	around end of buffer?
wrapmargin	wm	automatic line splitting

Scanning pattern formation

^	beginning of line
\$	end of line
.	any character
\<	beginning of word
\>	end of word
[str]	any char in str
[!str]	... not in str
[x-y]	... between x and y
*	any number of preceding

AUTHOR

Vi and ex are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

FILES

/usr/lib/ex?.?strings	error messages
/usr/lib/ex?.?recover	recover command
/usr/lib/ex?.?preserve	preserve command
/etc/termcap	describes capabilities of terminals
\$HOME/.exrc	editor startup file
./exrc	editor startup file
/tmp/Exnnnnnn	editor temporary
/tmp/Rxnnnnnn	named buffer temporary
/usr/preserve	preservation directory

SEE ALSO

awk(1), ed(1), grep(1), sed(1), vi(1), termcap(5)

CAVEATS AND BUGS

The *undo* command causes all marks to be lost on lines changed and then restored if the marked lines were changed.

Undo never clears the buffer modified condition.

The *z* command prints a number of logical rather than physical lines. More than a screen full of output may result if long lines are present.

File input/output errors don't print a name if the command line '-' option is used.

There is no easy way to do a single scan ignoring case.

The editor does not warn if text is placed in named buffers and not used before exiting the editor.

EX(1)

MUNIX

EX(1)

Null characters are discarded in input files, and cannot appear in resultant files.

THE UNIVERSITY OF CHICAGO PRESS

1900

NAME

`excr` - run a program on another system

SYNOPSIS

`excr system program arg ...`

DESCRIPTION

If the program name begins with a '/' then the program with that name on the named system will be run. However, if there is no '/' then the directories `"/bin"` and then `"/usr/bin"` on the named system are searched for the program. When it is found, the program is executed as though its root were on the named file system. If the normal `"exec"` returns `errno = ENOEXEC`, an attempt is made to execute the shell on the named file.

This provides the user with a different facility to the one provided by remote execution. Thus the command `"../U5/bin/who"` issued on system U1 will print out a list of the users on U1; to obtain a list of the users on U5 the `excr` command must be used - `"excr ../U5 who"`.

DIAGNOSTICS

An error message is printed if the program does not exist or cannot be executed.

FILES

`/usr/NCbin/excr`

The first part of the report is a general description of the project. It includes a statement of the purpose of the project, a description of the scope of the project, and a description of the methods used in the project. The second part of the report is a detailed description of the results of the project. It includes a description of the data collected, a description of the analysis of the data, and a description of the conclusions drawn from the data. The third part of the report is a discussion of the implications of the results of the project. It includes a discussion of the limitations of the study, a discussion of the strengths of the study, and a discussion of the future directions of the study.

The first part of the report is a general description of the project. It includes a statement of the purpose of the project, a description of the scope of the project, and a description of the methods used in the project. The second part of the report is a detailed description of the results of the project. It includes a description of the data collected, a description of the analysis of the data, and a description of the conclusions drawn from the data. The third part of the report is a discussion of the implications of the results of the project. It includes a discussion of the limitations of the study, a discussion of the strengths of the study, and a discussion of the future directions of the study.

The first part of the report is a general description of the project. It includes a statement of the purpose of the project, a description of the scope of the project, and a description of the methods used in the project. The second part of the report is a detailed description of the results of the project. It includes a description of the data collected, a description of the analysis of the data, and a description of the conclusions drawn from the data. The third part of the report is a discussion of the implications of the results of the project. It includes a discussion of the limitations of the study, a discussion of the strengths of the study, and a discussion of the future directions of the study.

100-100000

NAME

expand, unexpand - expand tabs to spaces, and vice versa

SYNOPSIS

```
expand [ -tabstop ] [ -tab1,tab2,...,tabn ] [ file ... ]  
unexpand [ -a ] [ file ... ]
```

DESCRIPTION

Expand processes the named files or the standard input writing the standard output with tabs changed into blanks. Backspace characters are preserved into the output and decrement the column count for tab calculations. *Expand* is useful for pre-processing character files (before sorting, looking at specific columns, etc.) that contain tabs.

If a single *tabstop* argument is given then tabs are set *tabstop* spaces apart instead of the default 8. If multiple tabstops are given then the tabs are set at those specific columns.

Unexpand puts tabs back into the data from the standard input or the named files and writes the result on the standard output. By default only leading blanks and tabs are reconverted to maximal strings of tabs. If the *-a* option is given, then tabs are inserted whenever they would compress the resultant file by replacing two or more characters.

THE UNIVERSITY OF CHICAGO

DEPARTMENT OF THE HISTORY OF ARTS

CHICAGO, ILLINOIS

TO THE HONORABLE THE PRESIDENT OF THE UNIVERSITY OF CHICAGO
FROM THE DEPARTMENT OF THE HISTORY OF ARTS
SUBJECT: A REPORT ON THE PROGRESS OF THE DEPARTMENT
DURING THE YEAR 1900-1901
The Department of the History of Arts has during the year 1900-1901
been engaged in a number of important projects. The first of these
has been the completion of the History of Art Survey, which was
begun in 1898. This survey has been a most successful one, and
has resulted in the publication of a number of valuable papers.
The second project has been the organization of a series of lectures
on the History of Art, which were given during the year. These
lectures have been most successful, and have resulted in a number
of valuable papers. The third project has been the organization of
a series of lectures on the History of Art, which were given during
the year. These lectures have been most successful, and have
resulted in a number of valuable papers. The fourth project has
been the organization of a series of lectures on the History of Art,
which were given during the year. These lectures have been most
successful, and have resulted in a number of valuable papers.

NAME

expr - evaluate arguments as an expression

SYNOPSIS

expr *arg* ...

DESCRIPTION

The arguments are taken as an expression. After evaluation, the result is written on the standard output. Each token of the expression is a separate argument.

The operators and keywords are listed below. The list is in order of increasing precedence, with equal precedence operators grouped.

expr | *expr*

yields the first *expr* if it is neither null nor '0', otherwise yields the second *expr*.

expr & *expr*

yields the first *expr* if neither *expr* is null or '0', otherwise yields '0'.

expr *relop* *expr*

where *relop* is one of < <= = != >= >, yields '1' if the indicated comparison is true, '0' if false. The comparison is numeric if both *expr* are integers, otherwise lexicographic.

expr + *expr*

expr - *expr*

addition or subtraction of the arguments.

expr * *expr*

expr / *expr*

expr % *expr*

multiplication, division, or remainder of the arguments.

expr : *expr*

The matching operator compares the string first argument with the regular expression second argument; regular expression syntax is the same as that of *ed*(1). The \(...\) pattern symbols can be used to select a portion of the first argument. Otherwise, the matching operator yields the number of characters matched ('0' on failure).

(*expr*)

parentheses for grouping.

Examples:

To add 1 to the Shell variable *a*:

a=`*expr* \$*a* + 1`

To find the filename part (least significant part) of the pathname stored in variable *a*, which may or may not contain '/':

expr \$*a* : '.*\/(.*)' '|' \$*a*

Note the quoted Shell metacharacters.

SEE ALSO

ed(1), *sh*(1), *test*(1)

DIAGNOSTICS

Expr returns the following exit codes:

- 0 if the expression is neither null nor '0',
- 1 if the expression is null or '0',
- 2 for invalid expressions.

NAME

f77 - Fortran 77 compiler

SYNOPSIS

f77 [option] ... file ...

DESCRIPTION

F77 is the UNIX Fortran 77 compiler. It accepts several types of arguments:

Arguments whose names end with '.f' are taken to be Fortran 77 source programs; they are compiled and linked. Each object program is left on the file in the current directory whose name is that of the source with '.o' substituted for '.f'; the executable file is left on 'a.out'.

Arguments whose names end with '.r' or '.e' are taken to be Ratfor or EFL source programs, respectively; these are first transformed by the appropriate preprocessor, then compiled by *f77*.

In the same way, arguments whose names end with '.c' or '.s' are taken to be C or assembly source programs and are compiled or assembled and linked.

Floating point is available in three different versions. The options are the same as in *cc*(1) and the description in *fp*(3) applies. When the Motorola Fast Floating Point Package is used, the compiler accepts double precision declarations, but handles them silently as single precision.

The following options have a similar meaning as in *cc*(1):

- c Suppress the linking phase.
- p Prepare object files for profiling, see *prof*(1).
- S Compile the named programs without linking, and leave the assembler-listing on corresponding files suffixed '.s'.
- o output
Name the final output file *output* instead of 'a.out'.
- f[FN]
Use Motorola Fast Floating Point (-f; default), use Motorola IEEE Floating Point Software (-fF), use FPP board with NSC 16081 processor (-fN).
- K n *n* is a number in C notation (e.g. 4096 or 0x1000 or 010000), which determines the stack size of the generated program (default 2048 bytes, cf. *stksiz*(1)).

The following options are peculiar to *f77*:

- onetrip
Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)
- 1 Same as '-onetrip'.
- U Do not convert UPPERCASE to lowercase.
- u Make the default type of a variable 'undefined' rather than using the default Fortran rules.

- C Compile code to check that subscripts are within declared array bounds.
- w Suppress all warning messages. If the option is '-w66', only Fortran 66 compatibility warnings are suppressed.
- Nx Change default sizes of certain tables. 'x' is one of the letters q,x,s,c or n followed directly by a decimal number. The defaults are:
 - Nq150 Maximum number of equivalence statements
 - Nx200 Maximum number of external definitions
 - Ns701 Maximum number of statement labels
 - Nc20 Maximum nesting of control structures
 - Nn401 Size of hash table
- l2 Use short integers (2 bytes) for Fortran type integer.
- l4 Use long integers (4 bytes) for Fortran type integer (default).
- ls Make subscripts short (default).
- ll Make subscripts long. This option must be given if an array or a common block is larger than 32 K bytes. It is possible to link subprograms compiled with -ls and -ll.
- F Apply EFL and Ratfor preprocessor to relevant files, put the result in the file with the suffix changed to '.f', but do not compile.
- m Apply the M4 preprocessor to each '.r' or '.e' file before transforming it with the Ratfor or EFL preprocessor.
- Ex Use the string *x* as an EFL option in processing '.e' files.
- Rx Use the string *x* as a Ratfor option in processing '.r' files.

Other arguments are taken to be either linker option arguments, or F77-compatible object programs, typically produced by an earlier run, or perhaps libraries of F77-compatible routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with name 'a.out'.

FILES

file.[fresc]	input file
file.o	object file
a.out	linked output
/usr/lib/f77pass1ffp	compiler pass for FFP
/usr/lib/f77pass1ieee	compiler pass for IEEE floating point
/lib/c1	pass 2
/lib/c2	pass 3
/usr/lib/libF77ffp.a	intrinsic function library FFP
/usr/lib/libF77mot.a	intrinsic function library Motorola IEEE
/usr/lib/libF77nsc.a	intrinsic function library NSC IEEE
/usr/lib/libl77*.a	Fortran I/O library also for ffp, mot, nsc
/lib/libffp.a	Floating point library FFP
/lib/libmot.a	Floating point library Motorola IEEE
/lib/libnsc.a	Floating point library NSC IEEE
/lib/libc.a	C library, see section 3

SEE ALSO

S. I. Feldman, P. J. Weinberger, *A Portable Fortran 77 Compiler*

cc(1), ld(1), prof(1)

DIAGNOSTICS

The diagnostics produced by *f77* itself are intended to be self-explanatory. Occasional messages may be produced by the linker.

BUGS

This compiler has run the Fortran Compiler Validation Suite FCVS 78 from the Federal Cobol Compiler Test Service (FCCTS). FCVS 78 is also known as "navytest". It passed all tests except two. One error is due to floating point accuracy. The other error is that a format statement must be given before an "assign"-statement that assigns the label of the format statement to an integer variable.

If a common block is defined in one source file and incompletely initialized in another one, the block is without warning truncated to the initialized size.

The following information is being furnished to you for your information and is not to be used for any other purpose.

The following information is being furnished to you for your information and is not to be used for any other purpose.

The following information is being furnished to you for your information and is not to be used for any other purpose.

NAME

factor - factor a number

SYNOPSIS

factor [number]

DESCRIPTION

When *factor* is invoked without an argument, it waits for a number to be typed in. If you type in a positive number less than 2^{50} (about 7.2×10^{16}) it will factor the number and print its prime factors; each one is printed the proper number of times. Then it waits for another number. It exits if it encounters a zero or any non-numeric character.

If *factor* is invoked with an argument, it factors the number as above and then exits.

Maximum time to factor is proportional to \sqrt{n} and occurs when n is prime or the square of a prime. It takes 1 minute to factor a prime near 10^{14} on a PDP-11.

DIAGNOSTICS

"Ouch" for input out of range or for garbage input.

14-00000

14-00000

14-00000

14-00000

14-00000

14-00000

14-00000

14-00000

14-00000

14-00000

NAME

file - determine file type

SYNOPSIS

file file ...

DESCRIPTION

File performs a series of tests on each argument in an attempt to classify it. If an argument appears to be ascii, *file* examines the first 512 bytes and tries to guess its language.

BUGS

It often makes mistakes. In particular it often suggests that command files are C programs.

10000

10000

10000

10000

10000

10000

10000

NAME

find - find files

SYNOPSIS

find [**-l** *maxlvl*] *path-name-list* *expression*

DESCRIPTION

Find recursively descends the directory hierarchy for each path name in the *path-name-list* (i.e., one or more path names) seeking files that match a boolean *expression* written in the primaries given below. If **-l** *maxlvl* is specified, the directory walk will be at most *maxlvl* levels deep. In the descriptions, the argument *n* is used as a decimal integer where **+n** means more than *n*, **-n** means less than *n* and *n* means exactly *n*.

-name *file* True if *file* matches the current file name. Normal shell argument syntax may be used if escaped (watch out for [, ? and *).

-perm *onum* True if the file permission flags exactly match the octal number *onum* (see *chmod*(1)). If *onum* is prefixed by a minus sign, more flag bits (017777, see *stat*(2)) become significant and the flags are compared:

(flags&onum)==onum

-type *c* True if the type of the file is *c*, where *c* is **b**, **c**, **d**, **p**, or **f** for block special file, character special file, directory, fifo (a.k.a named pipe), or plain file.

-links *n* True if the file has *n* links.

-user *uname* True if the file belongs to the user *uname*. If *uname* is numeric and does not appear as a login name in the */etc/passwd* file, it is taken as a user ID.

-group *gname* True if the file belongs to the group *gname*. If *gname* is numeric and does not appear in the */etc/group* file, it is taken as a group ID.

-size *n* True if the file is *n* blocks long (512 bytes per block).

-atime *n* True if the file has been accessed in *n* days.

-mtime *n* True if the file has been modified in *n* days.

-ctime *n* True if the file has been changed in *n* days.

-exec *cmd* True if the executed *cmd* returns a zero value as exit status. The end of *cmd* must be punctuated by an escaped semicolon. A command argument {} is replaced by the current path name.

-ok *cmd* Like **-exec** except that the generated command line is printed with a question mark first, and is executed only if the user responds by typing **y**.

-print Always true; causes the current path name to be printed.

-cpio *device* Write the current file on *device* in *cpio*(5) format (5120 byte records).

-newer *file* True if the current file has been modified more recently than the argument *file*.

(*expression*) True if the parenthesized expression is true (parentheses are special to the shell and must be escaped).

The primaries may be combined using the following operators (in order of decreasing precedence):

- 1) The negation of a primary (! is the unary *not* operator).
- 2) Concatenation of primaries (the *and* operation is implied by the juxtaposition of two primaries).
- 3) Alternation of primaries (-o is the *or* operator).

EXAMPLE

To remove all files named **a.out** or ***.o** that have not been accessed for a week:

```
find / \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
```

FILES

/etc/passwd, /etc/group

SEE ALSO

cpio(1), sh(1), test(1), stat(2), cpio(5), fs(5).

NAME

finger - user information lookup program

SYNOPSIS

finger [options] name ...

DESCRIPTION

By default *finger* lists the login name, full name, terminal name and write status (as a '*' before the terminal name if write permission is denied), idle time, login time, and office location and phone number (if they are known) for each current UNIX user. (Idle time is minutes if it is a single integer, hours and minutes if a ':' is present, or days and hours if a 'd' is present.)

A longer format also exists and is used by *finger* whenever a list of people's names is given. (Account names as well as first and last names of users are accepted.) This format is multi-line, and includes all the information described above as well as the user's home directory and login shell, any plan which the person has placed in the file *.plan* in their home directory, and the project on which they are working from the file *.project* also in the home directory.

Finger options include:

- m Match arguments only on user name.
- l Force long output format.
- p Suppress printing of the *.plan* files
- s Force short output format.

FILES

/etc/utmp	who file
/etc/passwd	for users names, offices, ...
/usr/adm/lastlog	last login times
~/.plan	plans
~/.project	projects

SEE ALSO

w(1), who(1)

AUTHOR

Earl T. Cohen

BUGS

Only the first line of the *.project* file is printed.

The encoding of the *gc* field is UCB dependent - it knows that an office '197MC' is '197M Cory Hall', and that '529BE' is '529B Evans Hall'.

A user information data base is in the works and will radically alter the way the information that *finger* uses is stored. *Finger* will require extensive modification when this is implemented.

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

1950

NAME

`fmt` - simple text formatter

SYNOPSIS

`fmt` [name ...]

DESCRIPTION

Fmt is a simple text formatter which reads the concatenation of input files (or standard input if none are given) and produces on standard output a version of its input with lines as close to 72 characters long as possible. The spacing at the beginning of the input lines is preserved in the output, as are blank lines and interword spacing.

Fmt is meant to format mail messages prior to sending, but may also be useful for other simple tasks. For instance, within visual mode of the *ex* editor (e.g. *vi*) the command

`!fmt`
will reformat a paragraph, evening the lines.

SEE ALSO

`nroff(1)`, `mail(1)`

AUTHOR

Kurt Shoens

BUGS

The program was designed to be simple and fast - for more complex operations, the standard text processors are likely to be more appropriate.

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

100

NAME

fold - fold long lines for finite width output device

SYNOPSIS

fold [-width] [file ...]

DESCRIPTION

Fold is a filter which will fold the contents of the specified files, or the standard input if no files are specified, breaking the lines to have maximum width *width*. The default for *width* is 80. *Width* should be a multiple of 8 if tabs are present, or the tabs should be expanded using *expand*(1) before coming to *fold*.

SEE ALSO

expand(1)

BUGS

If underlining is present it may be messed up by folding.

1000

1000

1000

1000

1000

NAME

fpr - printer filter for FORTRAN output

SYNOPSIS

fpr [file ...]

DESCRIPTION

Fpr is a printer filter for output of FORTRAN programs. The first character of each line is interpreted as a format control character. The following characters are recognized:

<space>

single line feed,

1 form feed,

0 double line feed,

+ no line feed, i.e. print over previous line.

Other characters are treated like a space, i.e. single line feed, also are totally empty lines.

Input is read from the files given as arguments or from standard input, a file name of - is also interpreted as standard input.

Output is written to standard output.

EXAMPLE

The command line

a.out | fpr | lpr

prints the output to channel 6 of the FORTRAN program a.out with format control on the line printer.

BUGS

It is not possible to print more than 66 lines per page due to *lpr* setting.

THE HISTORY OF THE UNITED STATES

The history of the United States is a story of growth and change. It is a story of a people who have built a great nation from a small colony.

The first settlers came to the United States in 1492. They were explorers who were looking for new lands to settle.

The first settlers were the Pilgrims. They came to the United States in 1620. They were looking for a place where they could practice their religion freely.

The Pilgrims were the first of many settlers who came to the United States. They were followed by the Puritans, the Quakers, and the Catholics.

The settlers who came to the United States were looking for a place where they could live in peace and harmony. They were looking for a place where they could build a better life for themselves and their children.

The settlers who came to the United States were looking for a place where they could be free. They were looking for a place where they could live without the oppression of a king or a government.

NAME

from - who is my mail from?

SYNOPSIS

from [-s sender] [user]

DESCRIPTION

From prints out the mail header lines in your mailbox file to show you who your mail is from. If *user* is specified, then *user*'s mailbox is examined instead of your own. If the -s option is given, then only headers for mail sent by *sender* are printed.

FILES

/usr/mail/*

SEE ALSO

mail(1), prmail(1)

NAME

get - get a version of an SCCS file

SYNOPSIS

get [-rSID] [-ccutoff] [-ilist] [-xlist] [-aseq-no.] [-k] [-e] [-l[p]]
[-p] [-m] [-n] [-s] [-b] [-g] [-t] file ...

DESCRIPTION

Get generates an ASCII text file from each named SCCS file according to the specifications given by its keyletter arguments, which begin with -. The arguments may be specified in any order, but all keyletter arguments apply to all named SCCS files. If a directory is named, *get* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with s.) and unreadable files are silently ignored. If a name of - is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed. Again, non-SCCS files and unreadable files are silently ignored.

The generated text is normally written into a file called the *g-file* whose name is derived from the SCCS file name by simply removing the leading s.; (see also *FILES*, below).

Each of the keyletter arguments is explained below as though only one SCCS file is to be processed, but the effects of any keyletter argument applies independently to each named file.

-rSID The SCCS *ID*entification string (SID) of the version (delta) of an SCCS file to be retrieved. Table 1 below shows, for the most useful cases, what version of an SCCS file is retrieved (as well as the SID of the version to be eventually created by *delta*(1) if the -e keyletter is also used), as a function of the SID specified.

-ccutoff *Cutoff* date-time, in the form:

YY[MM[DD[HH[MM[SS]]]]]

No changes (deltas) to the SCCS file which were created after the specified *cutoff* date-time are included in the generated ASCII text file. Units omitted from the date-time default to their maximum possible values; that is, -c7502 is equivalent to -c750228235959. Any number of non-numeric characters may separate the various 2 digit pieces of the *cutoff* date-time. This feature allows one to specify a *cutoff* date in the form: "-c77/2/2 9:22:25". Note that this implies that one may use the %E% and %U% identification keywords (see below) for nested *gets* within, say the input to a *send*(1C) command:

~!get "-c%E% %U%" s.file

-e Indicates that the *get* is for the purpose of editing or making a change (delta) to the SCCS file via a subsequent use of *delta*(1). The -e keyletter used in a *get* for a particular version (SID) of the SCCS file prevents further *gets* for editing on the same SID until *delta* is executed or the j (joint edit) flag is set in the SCCS file (see *admin*(1)). Concurrent use of *get* -e for different SIDs is always allowed.

If the *g-file* generated by *get* with an *-e* keyletter is accidentally ruined in the process of editing it, it may be regenerated by re-executing the *get* command with the *-k* keyletter in place of the *-e* keyletter.

SCCS file protection specified via the ceiling, floor, and authorized user list stored in the SCCS file (see *admin(1)*) are enforced when the *-e* keyletter is used.

- b** Used with the *-e* keyletter to indicate that the new delta should have an SID in a new branch as shown in Table 1. This keyletter is ignored if the *b* flag is not present in the file (see *admin(1)*) or if the retrieved *delta* is not a leaf *delta*. (A leaf *delta* is one that has no successors on the SCCS file tree.) Note: A branch *delta* may always be created from a non-leaf *delta*.
- ilist** A *list* of deltas to be included (forced to be applied) in the creation of the generated file. The *list* has the following syntax:

<list> ::= <range> | <list> , <range>
 <range> ::= SID | SID - SID

SID, the SCCS Identification of a delta, may be in any form shown in the "SID Specified" column of Table 1. Partial SIDs are interpreted as shown in the "SID Retrieved" column of Table 1.
- xlist** A *list* of deltas to be excluded (forced not to be applied) in the creation of the generated file. See the *-i* keyletter for the *list* format.
- k** Suppresses replacement of identification keywords (see below) in the retrieved text by their value. The *-k* keyletter is implied by the *-e* keyletter.
- l[p]** Causes a delta summary to be written into an *l-file*. If *-lp* is used then an *l-file* is not created; the delta summary is written on the standard output instead. See *FILES* for the format of the *l-file*.
- p** Causes the text retrieved from the SCCS file to be written on the standard output. No *g-file* is created. All output which normally goes to the standard output goes to file descriptor 2 instead, unless the *-s* keyletter is used, in which case it disappears.
- s** Suppresses all output normally written on the standard output. However, fatal error messages (which always go to file descriptor 2) remain unaffected.
- m** Causes each text line retrieved from the SCCS file to be preceded by the SID of the delta that inserted the text line in the SCCS file. The format is: SID, followed by a horizontal tab, followed by the text line.
- n** Causes each generated text line to be preceded with the %M% identification keyword value (see below). The format is: %M%

value, followed by a horizontal tab, followed by the text line. When both the `-m` and `-n` keyletters are used, the format is: `%M%` value, followed by a horizontal tab, followed by the `-m` keyletter generated format.

- `-g` Suppresses the actual retrieval of text from the SCCS file. It is primarily used to generate an *l-file*, or to verify the existence of a particular SID.
- `-t` Used to access the most recently created ("top") delta in a given release (e.g., `-r1`), or release and level (e.g., `-r1.2`).
- `-aseq-no.` The delta sequence number of the SCCS file delta (version) to be retrieved (see *sccsfile(5)*). This keyletter is used by the *comb(1)* command; it is not a generally useful keyletter, and users should not use it. If both the `-r` and `-a` keyletters are specified, the `-a` keyletter is used. Care should be taken when using the `-a` keyletter in conjunction with the `-e` keyletter, as the SID of the delta to be created may not be what one expects. The `-r` keyletter can be used with the `-a` and `-e` keyletters to control the naming of the SID of the delta to be created.

For each file processed, *get* responds (on the standard output) with the SID being accessed and with the number of lines retrieved from the SCCS file.

If the `-e` keyletter is used, the SID of the delta to be made appears after the SID accessed and before the number of lines generated. If there is more than one named file or if a directory or standard input is named, each file name is printed (preceded by a new-line) before it is processed. If the `-i` keyletter is used included deltas are listed following the notation "Included"; if the `-x` keyletter is used, excluded deltas are listed following the notation "Excluded".

TABLE 1. Determination of SCCS Identification String

SID* Specified	-b Keyletter Used†	Other Conditions	SID Retrieved	SID of Delta to be Created
none‡	no	R defaults to mR	mR.mL	mR.(mL+1)
none‡	yes	R defaults to mR	mR.mL	mR.mL.(mB+1).1
R	no	R > mR	mR.mL	R.1***
R	no	R = mR	mR.mL	mR.(mL+1)
R	yes	R > mR	mR.mL	mR.mL.(mB+1).1
R	yes	R = mR	mR.mL	mR.mL.(mB+1).1
R	-	R < mR and R does <i>not</i> exist	hR.mL**	hR.mL.(mB+1).1
R	-	Trunk succ.# in release > R and R exists	R.mL	R.mL.(mB+1).1
R.L	no	No trunk succ.	R.L	R.(L+1)
R.L	yes	No trunk succ.	R.L	R.L.(mB+1).1
R.L	-	Trunk succ. in release ≥ R	R.L	R.L.(mB+1).1

R.L.B	no	No branch succ.	R.L.B.mS	R.L.B.(mS+1)
R.L.B	yes	No branch succ.	R.L.B.mS	R.L.(mB+1).1
R.L.B.S	no	No branch succ.	R.L.B.S	R.L.B.(S+1)
R.L.B.S	yes	No branch succ.	R.L.B.S	R.L.(mB+1).1
R.L.B.S	-	Branch succ.	R.L.B.S	R.L.(mB+1).1

- * "R", "L", "B", and "S" are the "release", "level", "branch", and "sequence" components of the SID, respectively; "m" means "maximum". Thus, for example, "R.mL" means "the maximum level number within release R"; "R.L.(mB+1).1" means "the first sequence number on the *new* branch (i.e., maximum branch number plus one) of level L within release R". Note that if the SID specified is of the form "R.L", "R.L.B", or "R.L.B.S", each of the specified components *must* exist.
- ** "hR" is the highest *existing* release that is lower than the specified, *nonexistent*, release R.
- *** This is used to force creation of the *first* delta in a *new* release.
- # Successor.
- † The *-b* keyletter is effective only if the *b* flag (see *admin(1)*) is present in the file. An entry of *-* means "irrelevant".
- ‡ This case applies if the *d* (default SID) flag is *not* present in the file. If the *d* flag is present in the file, then the SID obtained from the *d* flag is interpreted as if it had been specified on the command line. Thus, one of the other cases in this table applies.

IDENTIFICATION KEYWORDS

Identifying information is inserted into the text retrieved from the SCCS file by replacing *identification keywords* with their value wherever they occur. The following keywords may be used in the text stored in an SCCS file:

Keyword	Value
%M%	Module name: either the value of the <i>m</i> flag in the file (see <i>admin(1)</i>), or if absent, the name of the SCCS file with the leading <i>s</i> . removed.
%I%	SCCS identification (SID) (%R%.%L%.%B%.%S%) of the retrieved text.
%R%	Release.
%L%	Level.
%B%	Branch.
%S%	Sequence.
%D%	Current date (YY/MM/DD).
%H%	Current date (MM/DD/YY).
%T%	Current time (HH:MM:SS).
%E%	Date newest applied delta was created (YY/MM/DD).
%G%	Date newest applied delta was created (MM/DD/YY).
%U%	Time newest applied delta was created (HH:MM:SS).
%Y%	Module type: value of the <i>t</i> flag in the SCCS file (see <i>admin(1)</i>).
%F%	SCCS file name.
%P%	Fully qualified SCCS file name.
%Q%	The value of the <i>q</i> flag in the file (see <i>admin(1)</i>).

- %CZ** Current line number. This keyword is intended for identifying messages output by the program such as "this shouldn't have happened" type errors. It is *not* intended to be used on every line to provide sequence numbers.
- %ZZ** The 4-character string @(#) recognizable by *what(1)*.
- %W%** A shorthand notation for constructing *what(1)* strings for UNIX program files. **%W%** = **%Z%****%M%**<horizontal-tab>**%I%**
- %A%** Another shorthand notation for constructing *what(1)* strings for non-UNIX program files. **%A%** = **%Z%****%Y%** **%M%** **%I%****%Z%**

FILES

Several auxiliary files may be created by *get*. These files are known generically as the *g-file*, *l-file*, *p-file*, and *z-file*. The letter before the hyphen is called the tag. An auxiliary file name is formed from the SCCS file name: the last component of all SCCS file names must be of the form *s.module-name*, the auxiliary files are named by replacing the leading *s* with the tag. The *g-file* is an exception to this scheme: the *g-file* is named by removing the *s*. prefix. For example, *s.xyz.c*, the auxiliary file names would be *xyz.c*, *l.xyz.c*, *p.xyz.c*, and *z.xyz.c*, respectively.

The *g-file*, which contains the generated text, is created in the current directory (unless the *-p* keyletter is used). A *g-file* is created in all cases, whether or not any lines of text were generated by the *get*. It is owned by the real user. If the *-k* keyletter is used or implied its mode is 644; otherwise its mode is 444. Only the real user need have write permission in the current directory.

The *l-file* contains a table showing which deltas were applied in generating the retrieved text. The *l-file* is created in the current directory if the *-l* keyletter is used; its mode is 444 and it is owned by the real user. Only the real user need have write permission in the current directory.

Lines in the *l-file* have the following format:

- a. A blank character if the delta was applied;
* otherwise.
- b. A blank character if the delta was applied or wasn't applied and ignored;
* if the delta wasn't applied and wasn't ignored.
- c. A code indicating a "special" reason why the delta was or was not applied:
"I": Included.
"X": Excluded.
"C": Cut off (by a *-c* keyletter).
- d. Blank.
- e. SCCS identification (SID).
- f. Tab character.
- g. Date and time (in the form YY/MM/DD HH:MM:SS) of creation.
- h. Blank.
- i. Login name of person who created *delta*.

The comments and **MR** data follow on subsequent lines, indented one horizontal tab character. A blank line terminates each entry.

The *p-file* is used to pass information resulting from a *get* with an *-e* keyletter along to *delta*. Its contents are also used to prevent a

subsequent execution of *get* with an *-e* keyletter for the same SID until *delta* is executed or the joint edit flag, *j*, (see *admin*(1)) is set in the SCCS file. The *p-file* is created in the directory containing the SCCS file and the effective user must have write permission in that directory. Its mode is 644 and it is owned by the effective user. The format of the *p-file* is: the gotten SID, followed by a blank, followed by the SID that the new delta will have when it is made, followed by a blank, followed by the login name of the real user, followed by a blank, followed by the date-time the *get* was executed, followed by a blank and the *-i* keyletter argument if it was present, followed by a blank and the *-x* keyletter argument if it was present, followed by a new-line. There can be an arbitrary number of lines in the *p-file* at any time; no two lines can have the same new delta SID.

The *z-file* serves as a *lock-out* mechanism against simultaneous updates. Its contents are the binary (2 bytes) process ID of the command (i.e., *get*) that created it. The *z-file* is created in the directory containing the SCCS file for the duration of *get*. The same protection restrictions as those for the *p-file* apply for the *z-file*. The *z-file* is created mode 444.

SEE ALSO

admin(1), *delta*(1), *help*(1), *prs*(1), *what*(1), *scsfile*(5).

Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

If the effective user has write permission (either explicitly or implicitly) in the directory containing the SCCS files, but the real user doesn't, then only one file may be named when the *-e* keyletter is used.

NAME

`getopt` — parse command options

SYNOPSIS

`set -- `getopt optstring $*``

DESCRIPTION

Getopt is used to break up options in command lines for easy parsing by shell procedures, and to check for legal options. *Optstring* is a string of recognized option letters (see `getopt(3C)`); if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. The special option `--` is used to delimit the end of the options. *Getopt* will place `--` in the arguments at the end of the options, or recognize it if used explicitly. The shell arguments (`$1 $2 . . .`) are reset so that each option is preceded by a `-` and in its own shell argument; each option argument is also in its own shell argument.

DIAGNOSTICS

Getopt prints an error message on the standard error when it encounters an option letter not included in *optstring*.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the options **a** and **b**, and the option **o**, which requires an argument.

```
set -- `getopt abo: $*`
if [ $? != 0 ]
then
    echo $USAGE
    exit 2
fi
for i in $*
do
    case $i in
        -a | -b)    FLAG=$i; shift;;
        -o)         OARG=$2;  shift; shift;;
        --)         shift; break;;
        esac
    done
```

This code will accept any of the following as equivalent:

```
cmd -aoarg file file
cmd -a -o arg file file
cmd -oarg -a file file
cmd -a -oarg -- file file
```

SEE ALSO

`sh(1)`, `getopt(3C)`.

NAME

graph - draw a graph

SYNOPSIS

graph [options]

DESCRIPTION

Graph with no options takes pairs of numbers from the standard input as abscissas and ordinates of a graph. Successive points are connected by straight lines. The graph is encoded on the standard output for display by the *tplot*(1G) filters.

If the coordinates of a point are followed by a non-numeric string, that string is printed as a label beginning on the point. Labels may be surrounded with quotes "", in which case they may be empty or contain blanks and numbers; labels never contain new-lines.

The following options are recognized, each as a separate argument:

- a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument (default 1). A second optional argument is the starting point for automatic abscissas (default 0 or lower limit given by -x).
- b Break (disconnect) the graph after each label in the input.
- c Character string given by next argument is default label for each point.
- g Next argument is grid style, 0 no grid, 1 frame with ticks, 2 full grid (default).
- l Next argument is label for graph.
- m Next argument is mode (style) of connecting lines: 0 disconnected, 1 connected (default). Some devices give distinguishable line styles for other small integers (e.g., the Tektronix 4014: 2=dotted, 3=dash-dot, 4=short-dash, 5=long-dash).
- s Save screen, don't erase before plotting.
- x [1] If 1 is present, x axis is logarithmic. Next 1 (or 2) arguments are lower (and upper) x limits. Third argument, if present, is grid spacing on x axis. Normally these quantities are determined automatically.
- y [1] Similarly for y .
- h Next argument is fraction of space for height.
- w Similarly for width.
- r Next argument is fraction of space to move right before plotting.
- u Similarly to move up before plotting.
- t Transpose horizontal and vertical axes. (Option -x now applies to the vertical axis.)

A legend indicating grid range is produced with a grid unless the -s option is present. If a specified lower limit exceeds the upper limit, the axis is reversed.

SEE ALSO

graphics(1G), spline(1G), tplot(1G).

BUGS

Graph stores all points internally and drops those for which there isn't room.

Segments that run out of bounds are dropped, not windowed.
Logarithmic axes may not be reversed.

NAME

grep, *egrep*, *fgrep* - search a file for a pattern

SYNOPSIS

```
grep [ options ] expression [ files ]
egrep [ options ] [ expression ] [ files ]
fgrep [ options ] [ strings ] [ files ]
```

DESCRIPTION

Commands of the *grep* family search the input *files* (standard input default) for lines matching a pattern. Normally, each line found is copied to the standard output. *Grep* patterns are limited regular *expressions* in the style of *ed*(1); it uses a compact non-deterministic algorithm. *Egrep* patterns are full regular *expressions*; it uses a fast deterministic algorithm that sometimes needs exponential space. *Fgrep* patterns are fixed *strings*; it is fast and compact. The following *options* are recognized:

- v All lines but those matching are printed.
- x (Exact) only lines matched in their entirety are printed (*fgrep* only).
- c Only a count of matching lines is printed.
- l Only the names of files with matching lines are listed (once), separated by new-lines.
- n Each line is preceded by its relative line number in the file.
- b Each line is preceded by the block number on which it was found. This is sometimes useful in locating disk block numbers by context.
- s The error messages produced for nonexistent or unreadable files are suppressed (*grep* only).
- e *expression*
Same as a simple *expression* argument, but useful when the *expression* begins with a - (does not work with *grep*).
- f *file*
The regular *expression* (*egrep*) or *strings* list (*fgrep*) is taken from the *file*.

In all cases, the file name is output if there is more than one input file. Care should be taken when using the characters *\$*, ***, *[*, *^*, *|*, *(*, *)*, and ** in *expression*, because they are also meaningful to the shell. It is safest to enclose the entire *expression* argument in single quotes '...'.

Fgrep searches for lines that contain one of the *strings* separated by new-lines.

Egrep accepts regular expressions as in *ed*(1), except for *\(* and *\)*, with the addition of:

1. A regular expression followed by *+* matches one or more occurrences of the regular expression.
2. A regular expression followed by *?* matches 0 or 1 occurrences of the regular expression.
3. Two regular expressions separated by *|* or by a new-line match strings that are matched by either.
4. A regular expression may be enclosed in parentheses *()* for grouping.

The order of precedence of operators is `[]`, then `*?+`, then concatenation, then `|` and new-line.

SEE ALSO

`ed(1)`, `sed(1)`, `sh(1)`.

DIAGNOSTICS

Exit status is 0 if any matches are found, 1 if none, 2 for syntax errors or inaccessible files (even if matches were found).

BUGS

Ideally there should be only one *grep*, but we don't know a single algorithm that spans a wide enough range of space-time tradeoffs. Lines are limited to 256 characters; longer lines are truncated. *Egrep* does not recognize ranges, such as `[a-z]`, in character classes.

NAME

head - give first few lines

SYNOPSIS

head [-count] [file ...]

DESCRIPTION

This filter gives the first *count* lines of each of the specified files, or of the standard input. If *count* is omitted it defaults to 10.

SEE ALSO

tail(1)

NAME

help - ask for help

SYNOPSIS

help [args]

DESCRIPTION

Help finds information to explain a message from a command or explain the use of a command. Zero or more arguments may be supplied. If no arguments are given, *help* will prompt for one.

The arguments may be either message numbers (which normally appear in parentheses following messages) or command names, of one of the following types:

- type 1 Begins with non-numeric, ends in numerics. The non-numeric prefix is usually an abbreviation for the program or set of routines which produced the message (e.g., *ge6*, for message 6 from the *get* command).
- type 2 Does not contain numerics (as a command, such as *get*)
- type 3 Is all numeric (e.g., 212)

The response of the program will be the explanatory information related to the argument, if there is any.

When all else fails, try "help stuck".

FILES

/usr/lib/help directory containing files of message text.

DIAGNOSTICS

Use *help*(1) for explanations.

1000
1000
1000

1000
1000
1000

1000
1000
1000

1000
1000
1000

1000
1000

1000
1000

NAME

hp - handle special functions of HP 2640 and 2621-series terminals

SYNOPSIS

hp [-e] [-m]

DESCRIPTION

Hp supports special functions of the Hewlett-Packard 2640 series of terminals, with the primary purpose of producing accurate representations of most *nroff*(1) output. Typical uses are:

```
nroff -h files ... | hp
nroff -h -s ... files | hp
```

In the latter case, *nroff* will stop at the beginning of each page (including the first) and wait for you to hit line-feed (control-j) before resuming output.

Regardless of the hardware options on your terminal, *hp* tries to do sensible things with underlining and reverse line-feeds. If the terminal has the "display enhancements" feature, subscripts and superscripts can be indicated in distinct ways. If it has the "mathematical-symbol" feature, Greek and other special characters can be displayed.

The flags are as follows:

- e It is assumed that your terminal has the "display enhancements" feature, and so maximal use is made of the added display modes. Overstruck characters are presented in the Underline mode. Superscripts are shown in Half-bright mode, and subscripts in Half-bright, Underlined mode. If this flag is omitted, *hp* assumes that your terminal lacks the "display enhancements" feature. In this case, all overstruck characters, subscripts, and superscripts are displayed in Inverse Video mode, i.e., dark-on-light, rather than the usual light-on-dark.
- m Requests minimization of output by removal of new-lines. Any contiguous sequence of 3 or more new-lines is converted into a sequence of only 2 new-lines; i.e., any number of successive blank lines produces only a single blank output line. This allows you to retain more actual text on the screen.

With regard to Greek and other special characters, *hp* provides the same set as does *300*(1), except that "not" is approximated by a right arrow, and only the top half of the integral sign is shown. The display is adequate for examining output from *neqn*(1).

DIAGNOSTICS

"line too long" if the representation of a line exceeds 1,024 characters. The exit codes are 0 for normal termination, 2 for all errors.

SEE ALSO

300(1), *col*(1), *greek*(1), *neqn*(1), *tbl*(1), *troff*(1).

BUGS

An "overstriking sequence" is defined as a printing character followed by a backspace followed by another printing character. In such sequences, if either printing character is an underscore, the other printing character is shown underlined or in Inverse Video; otherwise, only the first

printing character is shown (again, underlined or in Inverse Video). Nothing special is done if a backspace is adjacent to an ASCII control character. Sequences of control characters (e.g., reverse line-feeds, backspaces) can make text "disappear"; in particular, tables generated by *tbl(1)* that contain vertical lines will often be missing the lines of text that contain the "foot" of a vertical line, unless the input to *hp* is piped through *col(1)*.

Although some terminals do provide numerical superscript characters, no attempt is made to display them.

NAME

hyphen - find hyphenated words

SYNOPSIS

hyphen files

DESCRIPTION

Hyphen finds all the hyphenated words in *files* and prints them on the standard output. If no arguments are given, the standard input is used. Thus *hyphen* may be used as a filter.

BUGS

Hyphen can't cope with hyphenated *italic* (i.e., underlined) words; it will often miss them completely, or mangle them.

Hyphen occasionally gets confused, but with no ill effects other than spurious extra output.

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

100-10000

NAME

id - print user and group IDs and names

SYNOPSIS

id

DESCRIPTION

Id writes a message on the standard output giving the user and group IDs and the corresponding names of the invoking process. If the effective and real IDs do not match, both are printed.

SEE ALSO

logname(1), getuid(2), getgid(2).

100

100

100

100

NAME

ifprolog - The prolog interpreter system

SYNOPSIS

ifprolog [**-c** file] [**-ls** Size] [**-gs** Size] [**-ts** Size] [**-dbs** Size] [**-fs** Size] name ...

DESCRIPTION

ifprolog is a prolog interpreter written in C.

The options are:

-c File

The file File will be consulted, after the initialization of the interpreter. This option may occur several times.

-ls Size

A local stack area of Size bytes will be allocated.

-gs Size

A global stack area of Size bytes will be allocated.

-ts Size

A trail area of Size bytes will be allocated.

-dbs Size

A database area of Size bytes will be allocated.

-fs Size

A functor/atom area of Size bytes will be allocated.

SEE ALSO

Clocksin, W.F., Mellish, C.S.: Programming in Prolog; Springer Verlag Berlin Heidelberg 1981

IF/Prolog User's Manual, Interface Computer GmbH

LIMITATIONS

ifprolog needs at least 300 Kbyte memory

FILES

/usr/lib/ifprolog/pl/boot needed for initialisation

/usr/lib/ifprolog/pl/init needed for initialisation

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

NAME

indent – indent and format a C program source

SYNOPSIS

indent ifile [ofile] [args]

DESCRIPTION

The arguments that can be specified follows. They may appear before or after the file names.

- ifile Input file specification.
- ofile Output file specification.
If omitted, then the indented formatted file will be written back into the input file, and there will be a "back-up" copy of ifile written in the current directory. For an ifile named "/blah/blah/file", the backup file will be named ".Bfile". (It will only be listed when the '-a' argument is specified in ls.) If ofile is specified, indent checks to make sure it is different from ifile.
- lnnn Maximum length of an output line. The default is 75.
- cnnn The column in which comments will start. The default is 33.
- cdnnn The column in which comments on declarations will start. The default is for these comments to start in the same column as other comments.
- innn The number of spaces for one indentation level. The default is 4.
- dj,-ndj -dj will cause declarations to be left justified. -ndj will cause them to be indented the same as code. The default is -ndj.
- v,-nv -v turns on "verbose" mode, -nv turns it off. When in verbose mode, indent will report when it splits one line of input into two or more lines of output, and it will give some size statistics at completion. The default is -nv.
- bc,-nbc If -bc is specified, then a newline will be forced after each comma in a declaration. -nbc will turn off this option. The default is -bc.
- dnnn This option controls the placement of comments which are not to the right of code. Specifying -d2 means that such comments will be placed two indentation levels to the left of code. The default -d0 lines up these comments with the code. See the section on comment indentation below.
- br,-bl Specifying -bl will cause complex statements to be lined up like this:

```

        if (...)
        {
            code
        }

```

Specifying -br (the default) will make them look like this:

```

        if (...) {
            code
        }

```


You may set up your own 'profile' of defaults to indent by creating the file `'/usr/your-name/.indent.pro'` (where your-name is your login name) and including whatever switches you like. If indent is run and a profile file exists, then it is read to set up the program's defaults. Switches on the command line, though, will always over-ride profile switches. The profile file must be a single line of not more than 127 characters. The switches should be separated on the line by spaces or tabs. Indent is intended primarily as a C program indenter. Specifically, indent will:

- > indent code lines
- > align comments
- > insert spaces around operators where necessary
- > break up declaration lists as in `"int a,b,c;"`.

It will not break up long statements to make them fit within the maximum line length, but it will flag lines that are too long. Lines will be broken so that each statement starts a new line, and braces will appear alone on a line. (See the `-br` option to inhibit this.) Also, an attempt is made to line up identifiers in declarations.

Multi-line expressions

Indent will not break up complicated expressions that extend over multiple lines, but it will usually correctly indent such expressions which have already been broken up. Such an expression might end up looking like this:

```
x =
    (
        (Arbitrary parenthesized expression)
        +
        (
            (Parenthesized expression)
            *
            (Parenthesized expression)
        )
    );
```

Comments

Indent recognizes four kinds of comments. They are straight text, "box" comments, UNIX-style comments, and comments that should be passed thru unchanged. The action taken with these various types is as follows:

"Box" comments: The DSG documentation standards specify that boxes will be placed around section headers. Indent assumes that any comment with a dash immediately after the start of comment (i.e. `"/*-"`) is such a box. Each line of such a comment will be left unchanged, except that the first non-blank character of each successive line will be lined up with the beginning slash of the first line. Box comments will be indented (see below).

Unix-style comments: This is the type of section header which is used extensively in the UNIX system source. If the start of comment (`'/*'`)

appears on a line by itself, indent assumes that it is a UNIX-style comment. These will be treated similarly to box comments, except the first non-blank character on each line will be lined up with the '*' of the '/*'.

Unchanged comments: Any comment which starts in column 1 will be left completely unchanged. This is intended primarily for documentation header pages. The check for unchanged comments is made before the check for UNIX-style comments.

Straight text: All other comments are treated as straight text. Indent will fit as many words (separated by blanks, tabs, or newlines) on a line as possible. Straight text comments will be indented.

Comment indentation Box, UNIX-style, and straight text comments may be indented. If a comment is on a line with code it will be started in the "comment column", which is set by the -cnnn command line parameter. Otherwise, the comment will be started at nnn indentation levels less than where code is currently being placed, where nnn is specified by the -dnnn command line parameter. (Indented comments will never be placed in column 1.) If the code on a line extends past the comment column, the comment will be moved to the next line.

DIAGNOSTICS

Diagnostic error messages, mostly to tell that a text line has been broken or is too long for the output line, will be printed on the controlling tty.

FILES

/usr/your-name/.indent.pro - profile file

BUGS

Doesn't know how to format "long" declarations.

NAME

inews - submit news articles

SYNOPSIS

inews [-h] -t *title* [-n *newsgroups*] [-e *expiration date*]

inews -p [*filename*]

inews -C *newsgroup*

DESCRIPTION

Inews submits news articles to the USENET news network. It is intended as a raw interface, not as a human user interface. Casual users should probably use *postnews*(1) instead.

The first form is for submitting user articles. The body will be read from the standard input. A *title* must be specified as there is no default. Each article belongs to a list of newsgroups. If the -n flag is omitted, the list will default to something like *general*. (On ours, it is *general*.) If you wish to submit an article in multiple newsgroups, the *newsgroups* must be separated by commas and/or spaces. If not specified, the expiration date will be set to the local default. The -f flag specifies the article's sender. Without this flag, the sender defaults to the user's name. If -f is specified, the real sender's name will be included as a Sender line. The -h flag specifies that headers are present at the beginning of the article, and these headers should be included with the article header instead of as text. (This mechanism can be used to edit headers and supply additional nondefault headers, but not to specify certain information, such as the sender and article ID, that inews itself generates.)

When posting an article, the environment is checked for information about the sender. If NAME is found, its value is used for the full name, rather than the system value (often in /etc/passwd). This is useful if the system value cannot be set, or when more than one person uses the same login. If ORGANIZATION is found, the value overrides the system default organization. This is useful when a person uses a guest login and is not primarily associated with the organization owning the machine.

The second form is used for receiving articles from other machines. If *filename* is given, the article will be read from the specified file; otherwise the article will be read from the standard input. An expiration date need not be present and a receival date, if present, will be ignored.

After local installation, inews will transmit the article to all systems that subscribe to the newsgroups that the article belongs to.

The third form is for creating new newsgroups. On some systems, this may be limited to specific users such as the super-user or news administrator. (This happens on ours.)

If the file /usr/lib/news/recording is present, it is taken as a list of "recordings" to be shown to users posting news. (This is by analogy to the recording you hear when you dial information in some parts of the country, asking you if you really wanted to do this.) The file contains lines of the form:

newsgroups <tab> filename

for example:

net.all net.recording fa.all fa.recording

Any user posting an article to a newsgroup matching the pattern on the left will be shown the contents of the file on the right. The file is found in the LIB directory (often /usr/lib/news). The user is then told to hit DEL to abort or RETURN to proceed. The intent of this feature is to help companies keep proprietary information from accidentally leaking out.

FILES

/usr/spool/news/.sys.nnn	temporary articles
/usr/spool/news/newsgroups/article_no.	Articles
/usr/spool/oldnews/	Expired articles
/usr/lib/news/active	List of known newsgroups and highest local article numbers in each.
/usr/lib/news/seq	Sequence number of last article
/usr/lib/news/history	List of all articles ever seen
/usr/lib/news/sys	System subscription list

SEE ALSO

Mail(1), mail(1), getdate(3), msgs(1), news(5), newsrc(5), postnews(1), readnews(1), recnews(1), sendnews(8), uucp(1C), uurec(8).

AUTHORS

Matt Glickman
Mark Horton
Stephen Daniel
Tom R. Truscott

NAME

intro - introduction to commands

DESCRIPTION

This section describes publicly accessible commands in alphabetic order. Certain distinctions of purpose are made in the headings:

- (1) Commands of general utility.
- (1C) Commands for communication with other systems.
- (1G) Commands used primarily for graphics and computer-aided design.

System maintenance commands formerly in the category 1M have been moved to section 8.

SEE ALSO

Section (6) for computer games.

Section (8) for system maintenance programs.

How to get started, in the Introduction.

DIAGNOSTICS

Upon termination each command returns two bytes of status, one supplied by the system giving the cause for termination, and (in the case of 'normal' termination) one supplied by the program, see *wait* and *exit*(2). The former byte is 0 for normal termination, the latter is customarily 0 for successful execution, nonzero to indicate troubles such as erroneous parameters, bad or inaccessible data, or other inability to cope with the task at hand. It is called variously 'exit code', 'exit status' or 'return code', and is described only where special conventions are involved.

NAME

join - relational database operator

SYNOPSIS

join [options] file1 file2

DESCRIPTION

Join forms, on the standard output, a join of the two relations specified by the lines of *file1* and *file2*. If *file1* is '-', the standard input is used.

File1 and *file2* must be sorted in increasing ASCII collating sequence on the fields on which they are to be joined, normally the first in each line.

There is one line in the output for each pair of lines in *file1* and *file2* that have identical join fields. The output line normally consists of the common field, then the rest of the line from *file1*, then the rest of the line from *file2*.

Fields are normally separated by blank, tab or newline. In this case, multiple separators count as one, and leading separators are discarded.

These options are recognized:

- an In addition to the normal output, produce a line for each unpairable line in file *n*, where *n* is 1 or 2.
- e s Replace empty output fields by string *s*.
- jn m Join on the *m*th field of file *n*. If *n* is missing, use the *m*th field in each file.
- o list
Each output line comprises the fields specified in *list*, each element of which has the form *n.m*, where *n* is a file number and *m* is a field number.
- tc Use character *c* as a separator (tab character). Every appearance of *c* in a line is significant.

SEE ALSO

sort(1), comm(1), awk(1)

BUGS

With default field separation, the collating sequence is that of *sort -b*; with *-t*, the sequence is that of a plain sort.

The conventions of *join*, *sort*, *comm*, *uniq*, *look* and *awk(1)* are wildly incongruous.

Dear Mr. [Name],

I am writing to you regarding the [Topic] which was discussed at our meeting on [Date]. The [Topic] is of great importance to our organization and we are currently working on a plan to address it.

The plan involves [Description of Plan] and we are currently in the process of [Action Being Taken]. We are confident that this plan will be successful in addressing the [Topic] and we are looking forward to your feedback on the plan.

I am sure that you will find this plan to be a [Description of Plan] and we are looking forward to your feedback on the plan.

Very truly yours,

[Signature]

[Name]

NAME

kill - terminate a process

SYNOPSIS

kill [-signo] processid ...

DESCRIPTION

Kill sends signal 15 (terminate) to the specified processes. This will normally kill processes that do not catch or ignore the signal. The process number of each asynchronous process started with *&* is reported by the Shell (unless more than one process is started in a pipeline, in which case the number of the last process in the pipeline is reported). Process numbers can also be found by using *ps*(1).

The details of the kill are described in *kill*(2). For example, if process number 0 is specified, all processes in the process group are signaled.

The killed process must belong to the current user unless he is the super-user.

If a signal number preceded by - is given as first argument, that signal is sent instead of terminate (see *signal*(2)). In particular "kill -9 ..." is a sure kill.

SEE ALSO

ps(1), *sh*(1), *kill*(2), *signal*(2).

Section 100

The first section of the report deals with the general situation of the country. It is a very interesting and informative account of the country and its people. The author has done a great deal of research and has written a very well informed and interesting account of the country and its people.

The second section of the report deals with the history of the country. It is a very interesting and informative account of the history of the country and its people. The author has done a great deal of research and has written a very well informed and interesting account of the history of the country and its people.

The third section of the report deals with the present situation of the country. It is a very interesting and informative account of the present situation of the country and its people. The author has done a great deal of research and has written a very well informed and interesting account of the present situation of the country and its people.

NAME

ld - loader

SYNOPSIS

ld [option] file ...

DESCRIPTION

Ld combines several object programs into one, resolves external references, and searches libraries. In the simplest case several object *files* are given, and *ld* combines them, producing an object module which can be either executed or become the input for a further *ld* run. (In the latter case, the *-r* option must be given to preserve the relocation bits.) The output of *ld* is left on *a.out*. This file is made executable only if no errors occurred during the load.

The argument routines are concatenated in the order specified. If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, and the library has not been processed by *ranlib*(1), the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries may be important. If the first member of a library is named *'__SYMDEF'*, then it is understood to be a dictionary for the library such as produced by *ranlib*; the dictionary is searched iteratively to satisfy as many references as possible.

The symbols *'__etext'*, *'__edata'* and *'__end'* (*'etext'*, *'edata'* and *'end'* in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld understands several options. Except for *-l*, they should appear before the file names. They may be catenated in the usual form, e.g. *'ld -eso _start prog main.o'*. *Ld* without arguments records all valid options.

- 4* This option changes the library name *'/lib/libx.a'* to *'/lib/libLx.a'*. For example instead of */lib/libc.a* the library */lib/libLc.a* is searched. These libraries are assumed to include modules which have been produced with *cc -4*.
- b* This option is used to link Unix or standalone programs, where code is in segment 0 and data and bss in segment 1.
- d* Force definition of common storage even if the *-r* flag is present.
- e* The following argument is taken to be the name of the entry point of the loaded program. Default is the label *'__entry'* if it exists or otherwise location 0.
- lx* This option is an abbreviation for the library name *'/lib/libx.a'*, where *x* is a string. If that does not exist, *ld* tries *'/usr/lib/libx.a'*. A library is searched when its name is encountered, so the placement of a *-l* is significant.
- o* The *name* argument after *-o* is used as the name of the *ld* output file, instead of *a.out*.

- p This argument produces a symbol listing on stdout.
- r Generate relocation bits in the output file so that it can be the subject of another *ld* run. This flag also prevents final definitions from being given to common symbols, and suppresses the 'undefined symbol' diagnostics.
- s 'Strip' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by *strip*(1).
- t 'Trace' library searches: Print for every object module the reason why it is loaded.
- u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.
- x Do not preserve local (non-globl) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.
- y This argument links objects for an unmapped address range, such that code and data start from 0xff8000 and bss starts from 0xffa000. Useful for Macsbug etc.
- K The following argument is a number in C notation, which determines the stack size of the generated program (cf. *stksiz*(1)).
- O This is an overlay file, only the text segment will be replaced by *exec*(2). Shared data must have the same layout as in the program overlaid.
- Y The following argument is a number in C notation, which determines the size of the symbol table (default 1009 symbols). -Y should be the first ld option.

FILES

/lib/lib*.a	libraries
/lib/libL*.a	libraries with four-byte integers
/usr/lib/lib*.a	more libraries
a.out	output file

SEE ALSO

as(1), ar(1), cc(1), ranlib(1)

BUGS

Only 4-byte addresses are relocated; that is, a68 'addr:W' won't work.

The first part of the report is a general statement of the purpose and scope of the study. It is followed by a description of the methods used in the study, including the selection of subjects, the design of the study, and the procedures used to collect and analyze the data.

The second part of the report is a detailed description of the results of the study. This section includes a description of the data collected, a description of the statistical analysis performed, and a description of the results of the analysis. The results are presented in a series of tables and figures, which are used to illustrate the findings of the study.

The third part of the report is a discussion of the results of the study. This section includes a discussion of the implications of the findings, a discussion of the limitations of the study, and a discussion of the conclusions drawn from the study. The discussion is followed by a series of recommendations for further research.

The fourth part of the report is a conclusion. This section summarizes the findings of the study and provides a final statement of the conclusions drawn from the study. The conclusion is followed by a series of recommendations for further research.

The fifth part of the report is a list of references. This section lists the sources of information used in the study, including books, articles, and other documents. The references are listed in alphabetical order of the author's name.

NAME

learn - computer aided instruction about UNIX

SYNOPSIS

learn [-directory] [subject [lesson [speed]]]

DESCRIPTION

Learn gives CAI courses and practice in the use of UNIX. To get started simply type '*learn*'. The program will ask questions to find out what you want to do. The questions may be bypassed by naming a *subject*, and the last *lesson* number that *learn* told you in the previous session. You may also include a *speed* number that was given with the lesson number (but without the parentheses that *learn* places around the speed number). If *lesson* is '-', *learn* prompts for each lesson; this is useful for debugging.

The *subjects* presently handled are

- editor
- eqn
- files
- macros
- morefiles
- C

The special command 'bye' terminates a *learn* session.

The -*directory* option allows one to exercise a script in a nonstandard place.

FILES

/usr/lib/learn and all dependent directories and files

BUGS

The main strength of *learn*, that it asks the student to use the real UNIX, also makes possible baffling mistakes. It is helpful, especially for nonprogrammers, to have a UNIX initiate near at hand during the first sessions.

Occasionally lessons are incorrect, sometimes because the local version of a command operates in a non-standard way. Such lessons may be skipped, but it takes some sophistication to recognize the situation.

NAME

leave - remind you when you have to leave

SYNOPSIS

leave [hhmm]

DESCRIPTION

Leave waits until the specified time, then reminds you that you have to leave. You are reminded 5 minutes and 1 minute before the actual time, at the time, and every minute thereafter. When you log off, *leave* exits just before it would have printed the next message.

The time of day is in the form hhmm where hh is a time in hours (on a 12 or 24 hour clock). All times are converted to a 12 hour clock, and assumed to be in the next 12 hours.

If no argument is given, *leave* prompts with "When do you have to leave?". A reply of newline causes *leave* to exit, otherwise the reply is assumed to be a time. This form is suitable for inclusion in a *.login* or *.profile*.

Leave ignores interrupts, quits, and terminates. To get rid of it you should either log off or use "kill -9" giving its process id.

SEE ALSO

calendar(1)

AUTHOR

Mark Horton

BUGS

NAME

lex - generator of lexical analysis programs

SYNOPSIS

lex [-tvfn] [file] ...

DESCRIPTION

Lex generates programs to be used in simple lexical analysis of text. The input *files* (standard input default) contain regular expressions to be searched for, and actions written in C to be executed when expressions are found.

A C source program, 'lex.yy.c' is generated, to be compiled thus:

```
cc lex.yy.c -ll
```

This program, when run, copies unrecognized portions of the input to the output, and executes the associated C action for each regular expression that is recognized.

The following *lex* program converts upper case to lower, removes blanks at the end of lines, and replaces multiple blanks by single blanks.

```
%%  
[A-Z] putchar(yytext[0]+'a'-'A');  
[ ]+$  
[ ]+  putchar(' ');
```

The options have the following meanings.

- t Place the result on the standard output instead of in file 'lex.yy.c'.
- v Print a one-line summary of statistics of the generated analyzer.
- n Opposite of -v; -n is default.
- f 'Faster' compilation: don't bother to pack the resulting tables; limited to small programs.

SEE ALSO

yacc(1)

M. E. Lesk and E. Schmidt, *LEX - Lexical Analyzer Generator*

THE UNITED STATES OF AMERICA

1947

TO THE HONORABLE SENATE OF THE UNITED STATES

IN SENATE, FEBRUARY 11, 1947

REPORT OF THE COMMISSIONER OF THE GENERAL LAND OFFICE

ON THE LANDS OF THE UNITED STATES

AND

ON THE LANDS OF THE UNITED STATES

AND

ON THE LANDS OF THE UNITED STATES

AND

ON THE LANDS OF THE UNITED STATES

AND

ON THE LANDS OF THE UNITED STATES

AND

NAME

line - read one line

SYNOPSIS

line [lineno]

DESCRIPTION

Line copies one line (up to a new-line) from the standard input and writes it on the standard output. If no parameter is given, the first line is read. The command *line 2* will read the second line etc. It returns an exit code of 1 on EOF and always prints at least a new-line. It is often used within shell files to read from the user's terminal.

SEE ALSO

sh(1), read(2).

1000

1000

1000

1000

1000

1000

1000

1000

1000

NAME

lint - a C program verifier

SYNOPSIS

lint [-abchnpuvx] file ...

DESCRIPTION

Lint attempts to detect features of the C program *files* which are likely to be bugs, or non-portable, or wasteful. It also checks the type usage of the program more strictly than the compilers. Among the things which are currently found are unreachable statements, loops not entered at the top, automatic variables declared and not used, and logical expressions whose value is constant. Moreover, the usage of functions is checked to find functions which return values in some places and not in others, functions called with varying numbers of arguments, and functions whose values are not used.

By default, it is assumed that all the *files* are to be loaded together; they are checked for mutual compatibility. Function definitions for certain libraries are available to *lint*; these libraries are referred to by a conventional name, such as '-lm', in the style of *ld*(1).

Any number of the options in the following list may be used. The -D, -U, and -I options of *cc*(1) are also recognized as separate arguments.

- p Attempt to check portability to the *IBM* and *GCOS* dialects of C.
- h Apply a number of heuristic tests to attempt to intuit bugs, improve style, and reduce waste.
- b Report *break* statements that cannot be reached. (This is not the default because, unfortunately, most *lex* and many *yacc* outputs produce dozens of such comments.)
- v Suppress complaints about unused arguments in functions.
- x Report variables referred to by extern declarations, but never used.
- a Report assignments of long values to int variables.
- c Complain about casts which have questionable portability.
- u Do not complain about functions and variables used and not defined, or defined and not used (this is suitable for running *lint* on a subset of files out of a larger program).
- n Do not check compatibility against the standard library.

Exit(2) and other functions which do not return are not understood; this causes various lies.

Certain conventional comments in the C source will change the behavior of *lint*:

```
/*NOTREACHED*/
```

at appropriate points stops comments about unreachable code.

```
/*VARARGSn*/
```

suppresses the usual checking for variable numbers of arguments in the following function declaration. The data types of the first *n* arguments are checked; a missing *n* is taken to be 0.

/*NOSTRICT*/
shuts off strict type checking in the next expression.

/*ARGSUSED*/
turns on the **-v** option for the next function.

/*LINTLIBRARY*/
at the beginning of a file shuts off complaints about unused functions in this file.

FILES

/usr/lib/lint[12] programs
/usr/lib/lib-lc declarations for standard functions
/usr/lib/lib-port declarations for portable functions

SEE ALSO

cc(1)
S. C. Johnson, *Lint, a C Program Checker*

BUGS

Lint has yet to be made compatible with the current definition of C.

NAME

lock - reserve a terminal

SYNOPSIS

lock

DESCRIPTION

Lock requests a password from the user, then prints "LOCKED" on the terminal and refuses to relinquish the terminal until the password is repeated. If the user forgets the password, he has no other recourse but to login elsewhere and kill the lock process.

AUTHOR

Kurt Shoens

BUGS

Should timeout after 15 minutes.

NAME

login - sign on

SYNOPSIS

login [name [env-var ...]]

DESCRIPTION

The *login* command is used at the beginning of each terminal session and allows you to identify yourself to the system. It may be invoked as a command or by the system when a connection is first established. Also, it is invoked by the system when a previous user has terminated the initial shell by typing a *cntrl-z* to indicate an "end-of-file."

The shell handles login specially, so that login used as a command supercedes the shell, as if "exec login" had been typed.

Login asks for your user name (if not supplied as an argument), and, if appropriate, your password. Echoing is turned off (where possible) during the typing of your password, so it will not appear on the written record of the session.

At some installations, an option may be invoked that will require you to enter a second "dialup" password. This will occur only for dial-up connections, and will be prompted by the message "dialup password:". Both passwords are required for a successful login.

If you do not complete the login successfully within a certain period of time (e.g., one minute), you are likely to be silently disconnected.

After a successful login, accounting files are updated, the procedure */etc/profile* is performed, the message-of-the-day, if any, is printed, the user-ID, the group-ID, the working directory, and the command interpreter (usually *sh*(1)) is initialized, and the file *.profile* in the working directory is executed, if it exists. These specifications are found in the */etc/passwd* file entry for the user. The name of the command interpreter is - followed by the last component of the interpreter's pathname (i.e., *-sh*). If this field in the password file is empty, then the default command interpreter, */bin/sh* is used.

The basic *environment* (see *environ*(7)) is initialized to:

```
HOME=your-login-directory
PATH=/bin:/usr/bin
SHELL=last-field-of-passwd-entry
MAIL=/usr/mail/your-login-name
TZ=timezone-specification
```

The environment may be expanded or modified by supplying additional arguments to *login*, either at execution time or when *login* requests your login name. The arguments may take either the form *xxx* or *xxx=yyy*. Arguments without an equal sign are placed in the environment as

Ln=xxx

where *n* is a number starting at 0 and is incremented each time a new variable name is required. Variables containing an = are placed into the environment without modification. If they already appear in the environment, then they replace the older value. There are two exceptions. The variables *PATH* and *SHELL* cannot be changed. This prevents people,

logging into restricted shell environments, from spawning secondary shells which aren't restricted. Both *login* and *getty* understand simple single character quoting conventions. Typing a backslash in front of a character quotes it and allows the inclusion of such things as spaces and tabs.

FILES

/etc/utmp	accounting
/etc/wtmp	accounting
/usr/mail/ <i>your-name</i>	mailbox for user <i>your-name</i>
/etc/motd	message-of-the-day
/etc/passwd	password file
/etc/profile	system profile
.profile	user's login profile

SEE ALSO

mail(1), newgrp(1), sh(1), su(1), passwd(5), profile(5), environ(7).

DIAGNOSTICS

Login incorrect if the user name or the password cannot be matched.

No shell, cannot open password file, or no directory: consult a UNIX System programming counselor.

No utmp entry. You must exec "login" from the lowest level "sh". if you attempted to execute *login* as a command from other than the initial shell.

NAME

logname - get login name

SYNOPSIS

logname

DESCRIPTION

Logname returns the contents of the environment variable **\$LOGNAME**, which is set when a user logs into the system.

FILES

/etc/profile

SEE ALSO

env(1), login(1), logname(3X), environ(7).

NAME

look - find lines in a sorted list

SYNOPSIS

look [-df] string [file]

DESCRIPTION

Look consults a sorted *file* and prints all lines that begin with *string*. It uses binary search.

The options *d* and *f* affect comparisons as in *sort*(1):

d 'Dictionary' order: only letters, digits, tabs and blanks participate in comparisons.

f Fold. Upper case letters compare equal to lower case.

If no *file* is specified, */usr/dict/words* is assumed with collating sequence *-df*.

FILES

/usr/dict/words

SEE ALSO

sort(1), *grep*(1)

Dear Mr. [Name]

I am writing to you regarding the [Topic]

which you mentioned in your letter of [Date]

I have reviewed the matter and find that the [Information]

is in accordance with the [Policy]

and I am pleased to inform you that the [Result]

has been approved. I am enclosing the [Document]

for your information and the [Copy]

of the [Letter] for your records. I am sure that you will find this [Satisfactory]

Sincerely,
[Signature]

[Name]

NAME

lorder - find ordering relation for an object library

SYNOPSIS

lorder file ...

DESCRIPTION

The input is one or more object or library archive (see *ar(1)*) files. The standard output is a list of pairs of object file names, meaning that the first file of the pair refers to external identifiers defined in the second. The output may be processed by *tsort(1)* to find an ordering of a library suitable for one-pass access by *ld(1)*.

This brash one-liner intends to build a new library from existing '.o' files.

ar cr library `lorder *.o | tsort`

FILES

*symref, *symdef
nm(1), sed(1), sort(1), join(1)

SEE ALSO

tsort(1), ld(1), ar(1)

BUGS

The names of object files, in and out of libraries, must end with '.o'; non-sense results otherwise.

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

NAME

`lpctrl` - set options on the parallel line printer

SYNOPSIS

`lpctrl` [option ...]

DESCRIPTION

lpctrl sets certain I/O options on the parallel line printer. With no argument, it reports the current settings of the options. The option strings are selected from the following set:

`trans` switch to transparent mode

`-trans` switch off transparent mode

`cap` map lower case letters to upper case (capital mode)

`-cap` do not map letters

`line n` set page length to *n* (*n*>0).

`col n` set line length to *n* (*n*>0).

`indent n`

set indent to *n* (*n*≥0).

FILES

`/dev/lp`

SEE ALSO

`lpr(1)`, `ioctl(2)`, `lp(4)`

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

NAME

`lpr` - line printer spooler

SYNOPSIS

`lpr` [option ...] [name ...]

DESCRIPTION

Lpr causes the named files to be queued for printing on a line printer. If no names appear, the standard input is assumed; thus *lpr* may be used as a filter.

The following options may be given (each as a separate argument and in any order) before any file name arguments:

- c Makes a copy of the file to be sent before returning to the user.
- r Removes the file after sending it.
- m When printing is complete, reports that fact by *mail*(1).
- n Does not report the completion of printing by *mail*(1). This is the default option.

FILES

/etc/passwd	user's identification and accounting data.
/usr/lib/lpd	line printer daemon.
/usr/spool/lpd/*	spool area.

SEE ALSO

lpd(8).

Received of _____
the sum of _____
for _____

£ _____
Pounds
_____ Shillings
_____ Pence

Witness my hand and seal this _____ day of _____
19____
_____ Secretary

NAME

ls, ll, l, lr, lf, lx - list contents of directory

SYNOPSIS

ls [-ltasdrucifg] name ...

DESCRIPTION

For each directory argument, *ls* lists the contents of the directory; for each file argument, *ls* repeats its name and any other information requested. The output is sorted alphabetically by default. When no argument is given, the current directory is listed. When several arguments are given, the arguments are first sorted appropriately, but file arguments appear before directories and their contents. There are several options:

- l List in long format, giving mode, number of links, owner, size in bytes, and time of last modification for each file. (See below.) If the file is a special file the size field will instead contain the major and minor device numbers.
- t Sort by time modified (latest first) instead of by name, as is normal.
- a List all entries; usually filenames starting with a '.' are suppressed.
- s Give size in blocks, including indirect blocks, for each entry.
- d If argument is a directory, list only its name, not its contents (mostly used with -l to get status on directory).
- r Reverse the order of sort to get reverse alphabetic or oldest first as appropriate.
- u Use time of last access instead of last modification for sorting (-t) or printing (-l).
- c Use time of last modification to inode (mode, etc.) instead of last modification to file for sorting (-t) or printing (-l).
- i Print i-number in first column of the report for each file listed.
- f Force each argument to be interpreted as a directory and list the name found in each slot. This option turns off -l, -t, -s, and -r, and turns on -a; the order is the order in which entries appear in the directory.
- g Give group ID instead of owner ID in long listing.

The mode printed under the -l option contains 11 characters which are interpreted as follows: the first character is

- d if the entry is a directory;
- b if the entry is a block-type special file;
- c if the entry is a character-type special file;
- if the entry is a plain file.

The next 9 characters are interpreted as three sets of three bits each. The first set refers to owner permissions; the next to permissions to others in the same user-group; and the last to all others. Within each set the three characters indicate permission respectively to read, to write, or to execute the file as a program. For a directory, 'execute' permission

is interpreted to mean permission to search the directory for a specified file. The permissions are indicated as follows:

- r if the file is readable;
- w if the file is writable;
- x if the file is executable;
- if the indicated permission is not granted.

The group-execute permission character is given as s if the file has set-group-ID mode; likewise the user-execute permission character is given as S if the file has set-user-ID mode.

The last character of the mode (normally 'x' or '-') is t if the 1000 bit of the mode is on. See *chmod(1)* for the meaning of this mode.

When the sizes of the files in a directory are listed, a total count of blocks, including indirect blocks is printed.

LINKS TO SH

The names ll, l, lr, lf, lx should be linked to ls. Ll is the same as ls -l; l gives a compact listing of the names; lr goes recursively through the directories; lf puts a '*' after the names of executable files and a '/' after the names of directories; lx sorts names rowwise instead of columnwise.

FILES

/etc/passwd to get user ID's for 'ls -l'.
/etc/group to get group ID's for 'ls -g'.
/bin/ls, /bin/ll, /bin/l, /bin/lr, /bin/lf, /bin/lx

NAME

ltroff, vtroff - troff to the CANON LBP or the Versatec V80

SYNOPSIS

ltroff/vtroff [-F majorfont] [-123 minorfont] [-length] [-x]
troff arguments

DESCRIPTION

Ltroff/vtroff runs *troff*(1) sending its output through various programs to produce typeset output on the V80 for vtroff or the CANON LBP 10 for ltroff. The -l (lower case l) option causes the output to be split onto successive pages every *length* inches rather than the default 11".

The default font is a Hershey font. If some other font is desired you can give a -F argument and then the font name. This will place normal, italic and bold versions of the font on positions 1, 2, and 3. To place a font only on a single position, you can give an argument of the form -n and the minor font name. A .r will be added to the minor font name if needed. Thus "vtroff -ms paper" will set a paper in the Hershey font, while "vtroff -F nonie -ms paper" will set the paper in the (sans serif) nonie font. The -x option asks for exact simulation of photo-typesetter output. (I.e. using the width tables for the C.A.T. photo-typesetter)

FILES

/usr/lib/tmac/tmac.vcatdefault font mounts and bug fixes
/usr/lib/fontinfo/* fixes for other fonts
/usr/lib/vfont directory containing fonts

SEE ALSO

troff(1), vfont(5)

BUGS

The Versatec fonts with a resolution of 240 dots per inch are also used for the LBP 10 with a resolution of 300 dots per inch. Accordingly, the fonts are smaller than they should be on the LBP. There exist modified macro packages -manl, -msl, -mel for the LBP, that set the default point size to 12 rather than 10. The resulting output is normally satisfactory, but complicated papers produced e.g. with eqn may exhibit errors.

NAME

m4 - macro processor

SYNOPSIS

m4 [options] [files]

DESCRIPTION

M4 is a macro processor intended as a front end for Ratfor, C, and other languages. Each of the argument files is processed in order; if there are no files, or if a file name is -, the standard input is read. The processed text is written on the standard output.

The options and their effects are as follows:

- e Operate interactively. Interrupts are ignored and the output is unbuffered. Using this mode requires a special state of mind.
- s Enable line sync output for the C preprocessor (#line ...)
- B*int* Change the size of the push-back and argument collection buffers from the default of 4,096.
- H*int* Change the size of the symbol table hash array from the default of 199. The size should be prime.
- S*int* Change the size of the call stack from the default of 100 slots. Macros take three slots, and non-macro arguments take one.
- T*int* Change the size of the token buffer from the default of 512 bytes.

To be effective, these flags must appear before any file names and before any -D or -U flags:

-D*name*[=*val*]

Defines *name* to *val* or to null in *val*'s absence.

-U*name*

undefines *name*.

Macro calls have the form:

name(arg1,arg2, ..., argn)

The (must immediately follow the name of the macro. If a defined macro name is not followed by a (, it is deemed to have no arguments. Leading unquoted blanks, tabs, and new-lines are ignored while collecting arguments. Potential macro names consist of alphabetic letters, digits, and underscore _, where the first character is not a digit.

Left and right single quotes are used to quote strings. The value of a quoted string is the string stripped of the quotes.

When a macro name is recognized, its arguments are collected by searching for a matching right parenthesis. Macro evaluation proceeds normally during the collection of the arguments, and any commas or right parentheses which happen to turn up within the value of a nested call are as effective as those in the original input text. After argument collection, the value of the macro is pushed back onto the input stream and rescanned.

M4 makes available the following built-in macros. They may be redefined, but once this is done the original meaning is lost. Their values are null unless otherwise stated.

- define** the second argument is installed as the value of the macro whose name is the first argument. Each occurrence of **\$n** in the replacement text, where **n** is a digit, is replaced by the **n**-th argument. Argument 0 is the name of the macro; missing arguments are replaced by the null string; **\$#** is replaced by the number of arguments; **\$*** is replaced by a list of all the arguments separated by commas; **\$@** is like **\$***, but each argument is quoted (with the current quotes).
- undefine** removes the definition of the macro named in its argument.
- defn** returns the quoted definition of its argument(s). It is useful for renaming macros, especially built-ins.
- pushdef** like *define*, but saves any previous definition.
- popdef** removes current definition of its argument(s), exposing the previous one if any.
- ifdef** if the first argument is defined, the value is the second argument, otherwise the third. If there is no third argument, the value is null. The word *unix* is predefined on UNIX versions of *m4*.
- shift** returns all but its first argument. The other arguments are quoted and pushed back with commas in between. The quoting nullifies the effect of the extra scan that will subsequently be performed.
- changequote** change quote symbols to the first and second arguments. The symbols may be up to five characters long. *Changequote* without arguments restores the original values (i.e., ' ').
- changecom** change left and right comment markers from the default # and new-line. With no arguments, the comment mechanism is effectively disabled. With one argument, the left marker becomes the argument and the right marker becomes new-line. With two arguments, both markers are affected. Comment markers may be up to five characters long.
- divert** *m4* maintains 10 output streams, numbered 0-9. The final output is the concatenation of the streams in numerical order; initially stream 0 is the current stream. The *divert* macro changes the current output stream to its (digit-string) argument. Output diverted to a stream other than 0 through 9 is discarded.
- undivert** causes immediate output of text from diversions named as arguments, or all diversions if no argument. Text may be undiverted into another diversion. Undiverting discards the diverted text.
- divnum** returns the value of the current output stream.
- dnl** reads and discards characters up to and including the next new-line.

ifelse	has three or more arguments. If the first argument is the same string as the second, then the value is the third argument. If not, and if there are more than four arguments, the process is repeated with arguments 4, 5, 6 and 7. Otherwise, the value is either the fourth string, or, if it is not present, null.
incr	returns the value of its argument incremented by 1. The value of the argument is calculated by interpreting an initial digit-string as a decimal number.
decr	returns the value of its argument decremented by 1.
eval	evaluates its argument as an arithmetic expression, using 32-bit arithmetic. Operators include +, -, *, /, %, ^ (exponentiation), bitwise &, , ^, and ~; relationals; parentheses. Octal and hex numbers may be specified as in C. The second argument specifies the radix for the result; the default is 10. The third argument may be used to specify the minimum number of digits in the result.
len	returns the number of characters in its argument.
index	returns the position in its first argument where the second argument begins (zero origin), or -1 if the second argument does not occur.
substr	returns a substring of its first argument. The second argument is a zero origin number selecting the first character; the third argument indicates the length of the substring. A missing third argument is taken to be large enough to extend to the end of the first string.
translit	transliterates the characters in its first argument from the set given by the second argument to the set given by the third. No abbreviations are permitted.
include	returns the contents of the file named in the argument.
sinclude	is identical to <i>include</i> , except that it says nothing if the file is inaccessible.
syscmd	executes the UNIX command given in the first argument. No value is returned.
sysval	is the return code from the last call to <i>syscmd</i> .
maketemp	fills in a string of XXXXX in its argument with the current process ID.
m4exit	causes immediate exit from <i>m4</i> . Argument 1, if given, is the exit code; the default is 0.
m4wrap	argument 1 will be pushed back at final EOF; example: <i>m4wrap(^\cleanup()^)</i>
errprint	prints its argument on the diagnostic output file.
dumpdef	prints current names and definitions, for the named items, or for all if no arguments are given.

`traceon` with no arguments, turns on tracing for all macros (including built-ins). Otherwise, turns on tracing for named macros.

`traceoff` turns off trace globally and for any macros specified. Macros specifically traced by *traceon* can be untraced only by specific calls to *traceoff*.

SEE ALSO

The M4 Macro Processor by B. W. Kernighan and D. M. Ritchie.

NAME

mail, rmail - send mail to users or read mail

SYNOPSIS

mail [-epqr] [-f file]

mail [-t] persons

rmail [-t] persons

DESCRIPTION

Mail without arguments prints a user's mail, message-by-message, in last-in, first-out order. For each message, the user is prompted with a ?, and a line is read from the standard input to determine the disposition of the message:

<new-line>	Go on to next message.
+	Same as <new-line>.
d	Delete message and go on to next message.
p	Print message again.
-	Go back to previous message.
s [files]	Save message in the named <i>files</i> (<i>mbox</i> is default).
w [files]	Save message, without its header, in the named <i>files</i> (<i>mbox</i> is default).
m [persons]	Mail the message to the named <i>persons</i> (yourself is default).
q	Put undeleted mail back in the <i>mailfile</i> and stop.
EOT (control-d)	Same as q.
x	Put all mail back in the <i>mailfile</i> unchanged and stop.
!command	Escape to the shell to do <i>command</i> .
*	Print a command summary.

The optional arguments alter the printing of the mail:

- e causes mail not to be printed. An exit value of 0 is returned if the user has mail; otherwise, an exit value of 1 is returned.
- p causes all mail to be printed without prompting for disposition.
- q causes *mail* to terminate after interrupts. Normally an interrupt only causes the termination of the message being printed.
- r causes messages to be printed in first-in, first-out order.
- f*file* causes *mail* to use *file* (e.g., *mbox*) instead of the default *mailfile*.

When *persons* are named, *mail* takes the standard input up to an end-of-file (or up to a line consisting of just a .) and adds it to each *person's* *mailfile*. The message is preceded by the sender's name and a postmark. Lines that look like postmarks in the message, (i.e., "From ...") are preceded with a >. The -t option causes the message to be preceded by all *persons* the *mail* is sent to. A *person* is usually a user name recognized by *login*(1). If a *person* being sent mail is not recognized, or if *mail* is interrupted during input, the file *dead.letter* will be saved to allow editing and resending.

To denote a recipient on a remote system, prefix *person* by the system name and exclamation mark (see *uucp*(1C)). Everything after the first exclamation mark in *persons* is interpreted by the remote system. In particular, if *persons* contains additional exclamation marks, it can

denote a sequence of machines through which the message is to be sent on the way to its ultimate destination. For example, specifying *a!b!cde* as a recipient's name causes the message to be sent to user *b!cde* on system *a*. System *a* will interpret that destination as a request to send the message to user *cde* on system *b*. This might be useful, for instance, if the sending system can access system *a* but not system *b*, and system *a* has access to system *b*.

The *mailfile* may be manipulated in two ways to alter the function of *mail*. The *other* permissions of the file may be read-write, read-only, or neither read nor write to allow different levels of privacy. If changed to other than the default, the file will be preserved even when empty to perpetuate the desired permissions. The file may also contain the first line:

Forward to *person*

which will cause all mail sent to the owner of the *mailfile* to be forwarded to *person*. This is especially useful to forward all of a person's mail to one machine in a multiple machine environment.

Rmail only permits the sending of mail; *uucp*(1C) uses *rmail* as a security precaution. *Rmail* is a link to *mail*.

When a user logs in, the presence of mail, if any, is indicated. Also, notification is made if new mail arrives while using *mail*.

FILES

/etc/passwd	to identify sender and locate persons
/usr/mail/ <i>user</i>	incoming mail for <i>user</i> ; i.e., the <i>mailfile</i>
\$HOME/mbox	saved mail
\$MAIL	variable containing path name of <i>mailfile</i>
/tmp/ma*	temporary file
/usr/mail/*.lock	lock for mail directory
dead.letter	unmailable text

SEE ALSO

login(1), uucp(1C), write(1).

BUGS

Race conditions sometimes result in a failure to remove a lock file. After an interrupt, the next message may not be printed; printing may be forced by typing a *p*.

NAME

mail.nc - mail(1) using the Newcastle Connection instead of uucp.

DESCRIPTION

Changes: in "system!person", "system" is now recognised as the name of the root of a system in the Newcastle Connection distributed naming tree, not a uucp(1C) system name (especially, there are no embedded "!"s in "system").

Mail.nc spawns a single mail process on each remote system to distribute the mail to all recipients on that system (local mail behaves as usual). Thus

"mail.nc ../../unix1!dave ../../unix2!dave ../../unix1!fred" calls
"mail dave fred" on ../../unix1 and
"mail dave" on ../../unix2.

SEE ALSO

mail(1) for a full description of this command.

BUGS

Cannot forward messages to remote systems from within mail.

If execution of mail on a remote system does not succeed, it is difficult to know what messages have been sent.

Mail expects all "real" mail commands to live in files of the same path-name on all systems.

NAME

make - maintain, update, and regenerate groups of programs

SYNOPSIS

make [-f makefile] [-p] [-i] [-k] [-s] [-r] [-n] [-b] [-e] [-m] [-t]
[-q] [-d] [names]

DESCRIPTION

The following is a brief description of all options and some special names:

- f *makefile* Description file name. *Makefile* is assumed to be the name of a description file. A file name of - denotes the standard input. The contents of *makefile* override the built-in rules if they are present.
- p Print out the complete set of macro definitions and target descriptions.
- i Ignore error codes returned by invoked commands. This mode is entered if the fake target name .IGNORE appears in the description file.
- k Abandon work on the current entry, but continue on other branches that do not depend on that entry.
- s Silent mode. Do not print command lines before executing. This mode is also entered if the fake target name .SILENT appears in the description file.
- r Do not use the built-in rules.
- n No execute mode. Print commands, but do not execute them. Even lines beginning with an @ are printed.
- b Compatibility mode for old makefiles.
- e Environment variables override assignments within makefiles.
- t Touch the target files (causing them to be up-to-date) rather than issue the usual commands.
- d Debug mode. Print out detailed information on files and times examined.
- q Question. The *make* command returns a zero or non-zero status code depending on whether the target file is or is not up-to-date.
- .DEFAULT If a file must be made but there are no explicit commands or relevant built-in rules, the commands associated with the name .DEFAULT are used if it exists.
- .PRECIOUS Dependents of this target will not be removed when quit or interrupt are hit.
- .SILENT Same effect as the -s option.
- .IGNORE Same effect as the -i option.

Make executes commands in *makefile* to update one or more target names. Name is typically a program. If no -f option is present, *makefile*, *Makefile*, *s.makefile*, and *s.Makefile* are tried in order. If *makefile* is -,

the standard input is taken. More than one `-f` makefile argument pair may appear.

Make updates a target only if it depends on files that are newer than the target. All prerequisite files of a target are added recursively to the list of targets. Missing files are deemed to be out of date.

Makefile contains a sequence of entries that specify dependencies. The first line of an entry is a blank-separated, non-null list of targets, then a `:`, then a (possibly null) list of prerequisite files or dependencies. Text following a `:` and all following lines that begin with a tab are shell commands to be executed to update the target. The first line that does not begin with a tab or `#` begins a new dependency or macro definition. Shell commands may be continued across lines with the `<backslash><new-line>` sequence. Sharp (`#`) and new-line surround comments.

The following *makefile* says that `pgm` depends on two files `a.o` and `b.o`, and that they in turn depend on their corresponding source files (`a.c` and `b.c`) and a common file `incl.h`:

```
pgm: a.o b.o
    cc a.o b.o -o pgm
a.o: incl.h a.c
    cc -c a.c
b.o: incl.h b.c
    cc -c b.c
```

Command lines are executed one at a time, each by its own shell. A line is printed when it is executed unless the `-s` option is present, or the entry `.SILENT:` is in *makefile*, or unless the first character of the command is `@`. The `-n` option specifies printing without execution; however, if the command line has the string `$(MAKE)` in it, the line is always executed (see discussion of the `MAKEFLAGS` macro under *Environment*). The `-t` (touch) option updates the modified date of a file without executing any commands.

Commands returning non-zero status normally terminate *make*. If the `-i` option is present, or the entry `.IGNORE:` appears in *makefile*, or if the line specifying the command begins with `<tab><hyphen>`, the error is ignored. If the `-k` option is present, work is abandoned on the current entry, but continues on other branches that do not depend on that entry.

The `-b` option allows old makefiles (those written for the old version of *make*) to run without errors. The difference between the old version of *make* and this version is that this version requires all dependency lines to have a (possibly null) command associated with them. The previous version of *make* assumed if no command was specified explicitly that the command was null.

Interrupt and quit cause the target to be deleted unless the target depends on the special name `.PRECIOUS`.

Environment

The environment is read by *make*. All variables are assumed to be macro definitions and processed as such. The environment variables are processed before any makefile and after the internal rules; thus, macro

assignments in a makefile override environment variables. The `-e` option causes the environment to override the macro assignments in a makefile.

The **MAKEFLAGS** environment variable is processed by *make* as containing any legal input option (except `-f`, `-p`, and `-d`) defined for the command line. Further, upon invocation, *make* "invents" the variable if it is not in the environment, puts the current options into it, and passes it on to invocations of commands. Thus, **MAKEFLAGS** always contains the current input options. This proves very useful for "super-makes". In fact, as noted above, when the `-n` option is used, the command **\$(MAKE)** is executed anyway; hence, one can perform a *make -n* recursively on a whole software system to see what would have been executed. This is because the `-n` is put in **MAKEFLAGS** and passed to further invocations of **\$(MAKE)**. This is one way of debugging all of the makefiles for a software project without actually doing anything.

Macros

Entries of the form *string1* = *string2* are macro definitions. Subsequent appearances of **\$(string1[:subst1=[subst2]])** are replaced by *string2*. The parentheses are optional if a single character macro name is used and there is no substitute sequence. The optional *:subst1=subst2* is a substitute sequence. If it is specified, all non-overlapping occurrences of *subst1* in the named macro are replaced by *subst2*. Strings (for the purposes of this type of substitution) are delimited by blanks, tabs, new-line characters, and beginnings of lines. An example of the use of the substitute sequence is shown under *Libraries*.

Internal Macros

There are five internally maintained macros which are useful for writing rules for building targets.

- \$*** The macro **\$*** stands for the file name part of the current dependent with the suffix deleted. It is evaluated only for inference rules.
- \$@** The **\$@** macro stands for the full target name of the current target. It is evaluated only for explicitly named dependencies.
- \$<** The **\$<** macro is only evaluated for inference rules or the **.DEFAULT** rule. It is the module which is out of date with respect to the target (i.e., the "manufactured" dependent file name). Thus, in the **.c.o** rule, the **\$<** macro would evaluate to the **.c** file. An example for making optimized **.o** files from **.c** files is:

```
.c.o:
    cc -c -O $*.c
```

or:

```
.c.o:
    cc -c -O $<
```

- \$?** The **\$?** macro is evaluated when explicit rules from the makefile are evaluated. It is the list of prerequisites that are out of date with respect to the target; essentially, those modules which must be rebuilt.
- \$%** The **\$%** macro is only evaluated when the target is an archive library member of the form **lib(file.o)**. In this case, **\$@** evaluates to **lib** and

\$% evaluates to the library member, **file.o**.

Four of the five macros can have alternative forms. When an upper case D or F is appended to any of the four macros the meaning is changed to "directory part" for D and "file part" for F. Thus, **\$(@D)** refers to the directory part of the string **\$@**. If there is no directory part, The only macro excluded from this alternative form is **\$?**. The reasons for this are debatable.

Suffixes

Certain names (for instance, those ending with **.o**) have inferable prerequisites such as **.c**, **.s**, etc. If no update commands for such a file appear in *makefile*, and if an inferable prerequisite exists, that prerequisite is compiled to make the target. In this case, *make* has inference rules which allow building files from other files by examining the suffixes and determining an appropriate inference rule to use. The current default inference rules are:

```
.c .c~ .sh .sh~ .c.o .c~.o .c~.c .s.o .s~.o .y.o .y~.o .l.o .l~.o
.y.c .y~.c .l.c .c.a .c~.a .s~.a .h~.h
```

The internal rules for *make* are contained in the source file **rules.c** for the *make* program. These rules can be locally modified. To print out the rules compiled into the *make* on any machine in a form suitable for recompilation, the following command is used:

```
make -fp - 2>/dev/null </dev/null
```

The only peculiarity in this output is the **(null)** string which *printf*(3S) prints when handed a null string.

A tilde in the above rules refers to an SCCS file (see *sccsfile*(5)). Thus, the rule **.c~.o** would transform an SCCS C source file into an object file (**.o**). Because the **s**. of the SCCS files is a prefix it is incompatible with *make*'s suffix point-of-view. Hence, the tilde is a way of changing any file reference into an SCCS file reference.

A rule with only one suffix (i.e. **.c**) is the definition of how to build **x** from **x.c**. In effect, the other suffix is null. This is useful for building targets from only one source file (e.g., shell procedures, simple C programs).

Additional suffixes are given as the dependency list for **.SUFFIXES**. Order is significant; the first possible name for which both a file and a rule exist is inferred as a prerequisite.

The default list is:

```
.SUFFIXES: .o .c .y .l .s
```

Here again, the above command for printing the internal rules will display the list of suffixes implemented on the current machine. Multiple suffix lists accumulate; **.SUFFIXES:** with no dependencies clears the list of suffixes.

Inference Rules

The first example can be done more briefly:

```
pgm: a.o b.o
cc a.o b.o -o pgm
```



```
a.o b.o: incl.h
```

This is because *make* has a set of internal rules for building files. The user may add rules to this list by simply putting them in the *makefile*.

Certain macros are used by the default inference rules to permit the inclusion of optional matter in any resulting commands. For example, **CFLAGS**, **LFLAGS**, and **YFLAGS** are used for compiler options to *cc*(1), *lex*(1), and *yacc*(1) respectively. Again, the previous method for examining the current rules is recommended.

The inference of prerequisites can be controlled. The rule to create a file with suffix *.o* from a file with suffix *.c* is specified as an entry with *.c.o*: as the target and no dependents. Shell commands associated with the target define the rule for making a *.o* file from a *.c* file. Any target that has no slashes in it and starts with a dot is identified as a rule and not a true target.

Libraries

If a target or dependency name contains parenthesis, it is assumed to be an archive library, the string within parenthesis referring to a member within the library. Thus *lib(file.o)* and *\$(LIB)(file.o)* both refer to an archive library which contains *file.o*. (This assumes the **LIB** macro has been previously defined.) The expression *\$(LIB)(file1.o file2.o)* is not legal. Rules pertaining to archive libraries have the form *XX.a* where the *XX* is the suffix from which the archive member is to be made. An unfortunate byproduct of the current implementation requires the *XX* to be different from the suffix of the archive member. Thus, one cannot have *lib(file.o)* depend upon *file.o* explicitly. The most common use of the archive interface follows. Here, we assume the source files are all C type source:

```
lib:  lib(file1.o) lib(file2.o) lib(file3.o)
      @echo lib is now up to date
.c.a:
      $(CC) -c $(CFLAGS) $<
      ar rv $@ $*.o
      rm -f $*.o
```

In fact, the *.c.a* rule listed above is built into *make* and is unnecessary in this example. A more interesting, but more limited example of an archive library maintenance construction follows:

```
lib:  lib(file1.o) lib(file2.o) lib(file3.o)
      $(CC) -c $(CFLAGS) $(?:.o=.c)
      ar rv lib $?
      rm $? @echo lib is now up to date
.c.a;
```

Here the substitution mode of the macro expansions is used. The **\$(?)** list is defined to be the set of object file names (inside *lib*) whose C source files are out of date. The substitution mode translates the *.o* to *.c*. (Unfortunately, one cannot as yet transform to *.c~*; however, this may become possible in the future.) Note also, the disabling of the *.c.a*: rule, which would have created each object file, one by one. This particular construct speeds up archive library maintenance considerably. This type of construct becomes very cumbersome if the archive library contains a

mix of assembly programs and C programs.

FILES

[Mm]akefile
s.[Mm]akefile

SEE ALSO

sh(1).

Make—A Program for Maintaining Computer Programs by S. I. Feldman.
An Augmented Version of Make by E. G. Bradford.

BUGS

Some commands return non-zero status inappropriately; use `-i` to overcome the difficulty. Commands that are directly executed by the shell, notably `cd(1)`, are ineffectual across new-lines in *make*. The syntax `(lib(file1.o file2.o file3.o))` is illegal. You cannot build `lib(file.o)` from `file.o`. The macro `$(a.o=c~)` doesn't work.

NAME

`man`, `manprog` - print entries in this manual

SYNOPSIS

`man` [options] [section] titles
 /usr/lib/manprog file

DESCRIPTION

Man locates and prints the entry of this manual named *title* in the specified *section*. (For historical reasons, the word "page" is often used as a synonym for "entry" in this context.) The *title* is entered in lower case. The *section* number may not have a letter suffix. If no *section* is specified, the whole manual is searched for *title* and all occurrences of it are printed. *Options* and their meanings are:

- t Typeset the entry in the default format (8.5"x11").
- s Typeset the entry in the small format (6"x9").
- T4014 Display the typeset output on a Tektronix 4014 terminal using *tc*(1).
- Ttek Same as -T4014.
- Tst Print the typeset output on the MHCC STARE facility (this option is not usable on most systems).
- Tvp Print the typeset output on a Versatec printer; this option is not available at all UNIX System sites.
- Tterm Format the entry using *nroff* and print it on the standard output (usually, the terminal); *term* is the terminal type (see *term*(7) and the explanation below); for a list of recognized values of *term*, type *help term*2. The default value of *term* is 450.
- w Print on the standard output only the *path names* of the entries, relative to /usr/man, or to the current directory for -d option.
- d Search the current directory rather than /usr/man; requires the full file name (e.g., *cu.1c*, rather than just *cu*).
- 12 Indicates that the manual entry is to be produced in 12-pitch. May be used when *\$TERM* (see below) is set to one of 300, 300s, 450, and 1620. (The pitch switch on the DASI 300 and 300s terminals must be manually set to 12 if this option is used.)
- c Causes *man* to invoke *col*(1); note that *col*(1) is invoked automatically by *man* unless *term* is one of 300, 300s, 450, 37, 4000a, 382, 4014, tek, 1620, and X.
- y Causes *man* to use the non-compacted version of the macros.

The above *options* other than -d, -c, and -y are mutually exclusive, except that the -s option may be used in conjunction with the first four -T options above. Any other *options* are passed to *troff*, *nroff*, or the *man*(7) macro package.

When using *nroff*, *man* examines the environment variable *\$TERM* (see *environ*(7)) and attempts to select options to *nroff*, as well as filters, that adapt the output to the terminal being used. The -Tterm option overrides the value of *\$TERM*; in particular, one should use -Tlp when sending the output of *man* to a line printer.

Section may be changed before each *title*.

As an example:

man man

would reproduce on the terminal this entry, as well as any other entries named *man* that may exist in other sections of the manual, e.g., *man*(7).

If the first line of the input for an entry consists solely of the string:

" \" x

where *x* is any combination of the three characters *c*, *e*, and *t*, and where there is exactly one blank between the double quote (") and *x*, then *man* will preprocess its input through the appropriate combination of *cw*(1), *eqn*(1) (*neqn* for *nroff*) and *tbl*(1), respectively; if *eqn* or *neqn* are invoked, they will automatically read the file */usr/pub/eqnchar* (see *eqnchar*(7)).

The *man* command executes *manprog* that takes a file name as its argument. *Manprog* calculates and returns a string of three register definitions used by the formatters identifying the date the file was last modified. The returned string has the form:

-rdday -rmmonth -ryyear

and is passed to *nroff* which sets this string as variables for the *man* macro package. Months are given from 0 to 11, therefore month is always 1 less than the actual month. The *man* macros calculate the correct month. If the *man* macro package is invoked as an option to *nroff/troff* (i.e., *nroff -man file*), then the current day/month/year is used as the printed date.

FILES

<i>/usr/man/man[1-8]/*</i>	the <i>UNIX System User's Manual</i>
<i>/usr/man/local/man[1-8]/*</i>	local additions
<i>/usr/lib/manprog.</i>	calculates modification dates of entries

SEE ALSO

cw(1), *eqn*(1), *nroff*(1), *tbl*(1), *tc*(1), *troff*(1), *environ*(7), *man*(7), *term*(7).

BUGS

All entries are supposed to be reproducible either on a typesetter or on a terminal. However, on a terminal some information is necessarily lost.

NAME

`man` - print sections of this manual

SYNOPSIS

`man` [option ...] [chapter] title ...

DESCRIPTION

Man locates and prints the section of this manual named *title* in the specified *chapter*. (In this context, the word 'page' is often used as a synonym for 'section'.) The *title* is entered in lower case. The *chapter* number does not need a letter suffix. If no *chapter* is specified, the whole manual is searched for *title* and all occurrences of it are printed.

Options and their meanings are:

- t Phototypeset the section using *troff*(1).
- n Print the section on the standard output using *nroff*(1).
- k Display the output on a Tektronix 4014 terminal using *troff*(1) and *tc*(1).
- e Appended or prefixed to any of the above causes the manual section to be preprocessed by *neqn* or *eqn*(1); -e alone means -te.
- w Print the path names of the manual sections, but do not print the sections themselves.

(default)

Copy an already formatted manual section to the terminal, or, if none is available, act as -n. It may be necessary to use a filter to adapt the output to the particular terminal's characteristics.

Further *options*, e.g. to specify the kind of terminal you have, are passed on to *troff*(1) or *nroff*. *Options* and *chapter* may be changed before each *title*.

For example:

`man man`

would reproduce this section, as well as any other sections named *man* that may exist in other chapters of the manual, e.g. *man*(7).

FILES

`/usr/man/man?/*`
`/usr/man/cat?/*`

SEE ALSO

nroff(1), *eqn*(1), *tc*(1), *man*(7), *catman*(8)

BUGS

The manual is supposed to be reproducible either on a phototypesetter or on a terminal. However, on a terminal some information is necessarily lost.

12/10/88

12/10/88

12/10/88

12/10/88

12/10/88

12/10/88

12/10/88

12/10/88

12/10/88

12/10/88

12/10/88

12/10/88

12/10/88

12/10/88

NAME

med - screen editor

SYNOPSIS

med file [startline] [searchkey]

med -

med

DESCRIPTION

Med calls the MED editor, which allows you to edit a file using the screen of your terminal and the cursor keys, somewhat like paper, pencil and eraser.

If you call the editor by *med file*, the first lines of the file will be shown on the terminal screen. If the *file* does not exist, it will be created.

Med called with no arguments continues where the previous *med* session ended.

Med called with argument "-" restores only one file from the last session.

The purpose of a screen editor is to create a new file, or to look at and change existing files. *Med* allows you to look through a file by moving a "window" (the text shown at the screen) over the files contents. The window may be moved up, down, to the left or the right. The "cursor" (a blinking underline or box) shows where *med* is at the moment. Initially *med* is in "replace mode", i.e. the characters typed will replace those under the cursor. The cursor may be moved around with the arrow keys `↑` `↓` `←` `→` or with `return`, `home` or `tab`.

To change text on the screen, the cursor has to be placed there and the new characters entered. `delch` deletes the character under the cursor. To insert characters in the middle of a word (like a 'd' in 'middle'), press `insert`; then new characters will be inserted, and the rest of the line will move to the right. Pressing `insert` again, *med* is switched back to "replace mode".

"Function keys" are special keys along the top or the right of the keyboard. Each function key does one job, such as moving some lines, looking for a word, or using another file. A picture of all the different function keys for your terminal should be attached to this description.

Some functions take arguments such as a number, a search string, a file name, etc. To enter an argument, type the sequence:

`enter` number or string `function`.

`enter` leaves an '@' marker on the screen to remind you where the cursor was. It then goes to the bottom line on the screen to read the argument. This bottom line is also used to display error messages.

Function keys and their meaning

Move around:

↑ ↓ → ←	Move cursor
±tab	Skip to next/previous tab stop
±page	Move to next/previous page
±line	Move a halfpage up/down
±search	Search for a word
goto	Goto line <i>n</i>
left	Move window left
right	Move window right

Use another file or window:

use	Use another file
window	Create/delete a window
chwin	Go to next window

Cut and paste:

backspace	Delete character left of cursor
close	Delete line(s) into CLOSE buffer
delch	Delete character
open	Insert blank line(s)
pick	Copy text to PICK buffer
put	Get text from PICK buffer
restore	Get text from CLOSE buffer

Others:

do	Run a Unix command
enter	Enter a parameter
exit	Go back to Unix
insert	Flip insert mode
quote	Control character escape
refresh	Refresh
save	Write changes to disk
chtabs	Change tabs
macro	collect keystrokes

enter *n* **↑ ↓ → ←**Move the cursor *n* lines/columns in the direction of the arrow.

`backspace`

Remove the last character entered; thus typing "E R X `backspace` R" is the same as "E R R". This also works in insert mode.

`ctab``enter` `ctab`

Set a tab stop at the current position.
Remove the tab stop at the current position.

`chwin``enter` `n` `chwin``enter` `↑↓←→` `chwin`

Go to the next window.
Go to window `n`. Windows are numbered in the order they are created.
Argument defined by the cursor is used as window number.

`close``enter` `close``enter` `n` `close``enter` `↑↓←→` `close`

Delete one line. It is saved in the CLOSE buffer.
Delete the rest of the line, replacing it with the line below.
Delete `n` lines and save them in the CLOSE buffer.
Delete lines or a rectangle defined by the cursor and put it into the CLOSE buffer.

`delch`

Delete the character at the current cursor position, moving the following characters to the left.

`do``enter` `do``enter` `cmd` `do`

Run the previous DO command exactly as it was given.
Take the current line as a command for the shell. Results are inserted below the current line.
`cmd` is a Unix command in the format `[n[l]] prg [arg...]` (in shell notation). It replaces `n` paragraphs (or `n` lines if `l` appears) by the result of running filter `prg` on that text with given `args`. The replaced paragraphs are saved in the CLOSE buffer.

`exit``enter` `a` `exit``enter` `ad` `exit`

Exit `med`, writing back the changed files to disk (a backup file named `*.bak` is created).
Exit, but do not save files.
Terminate with core dump.

`goto``enter` `goto``enter` `n` `goto``enter` `↑↓←→` `goto`

Move window to top of file.
Move window to end of file.
Move to the `n`th line.
Argument defined by the cursor is used as line number.

`insert`

Put the editor into insert mode. Subsequent characters are inserted at the cursor, i.e., characters to the right of the cursor are not replaced but moved to the right. Pressing `insert` again will place `med` back to replace mode.


```
left
enter left
enter n left
```

Move window left (about a 1/3 window width).
Make the current column the last one (if possible).
Move window n columns left.

```
+line
enter +line
enter n +line
```

Move window down (about 1/3 page).
Make the current line the top one.
Move window down n lines.

```
-line
enter -line
enter n -line
```

Move window up (about 1/3 page).
Make the current line the bottom one (if possible).
Move window up n lines.

```
open
enter open
enter n open
enter ↑↓↔→ open
```

Open up one blank line.
Split the line exactly at cursor position.
Open up for n blank lines.
Insert blank lines or rectangle in area defined by cursor.

```
+page
enter n +page
```

Move window down one page (page = size of window).
Move down n pages.

```
-page
enter n -page
```

Move window up one page.
Move up n pages.

```
pick
enter n pick
enter ↑↓↔→ pick
```

Put the current line into the PICK buffer.
Put n lines into the PICK buffer.
Place lines or rectangle defined by cursor in PICK buffer.

```
put
enter n put
enter ↑↓↔→ put
```

Insert the contents of the PICK buffer (i.e. the lines or rectangle last "picked") at the current position.
Insert n copies of the PICK buffer at the current position.
Argument defined by the cursor is used as number of copies.

```
quote
```

To put a control character into the file. `quote` echoes as a @, and whatever key on the keyboard you press next appears as some printable character. The two characters now behave as two characters on the screen, but they are really the single control character in the file. Changing the second letter changes the control character; changing the preceding mark results in two ordinary characters.

```
refresh
```

Redraw terminal display.

```
restore
```

Insert the contents of the CLOSE buffer (i.e. the lines or rectangle last deleted). Insertion is done at the current position.

```
enter n restore
```

Insert n copies of the CLOSE buffer at the current posi-

<code>enter</code> $\uparrow\downarrow\leftrightarrow$ <code>restore</code>	tion. Argument defined by the cursor is used as number of copies.
<code>right</code>	Move window right (about a 1/3 window width).
<code>enter</code> <code>right</code>	Make current column the first one.
<code>enter</code> <code>n</code> <code>right</code>	Move window right <code>n</code> columns.
<code>save</code>	Save the file shown in the current window.
<code>enter</code> <code>name</code> <code>save</code>	Save the file shown in the current window under filename <code>name</code> .
<code>enter</code> $\uparrow\downarrow\leftrightarrow$ <code>save</code>	Argument defined by the cursor is used as filename.
<code>+search</code>	Search forwards (from the cursor towards the end of the file) for an occurrence of the search key last used.
<code>enter</code> <code>+search</code>	Search key used is from cursor position up to next blank.
<code>enter</code> <code>word</code> <code>+search</code>	Search for the next occurrence of <code>word</code> .
<code>enter</code> $\uparrow\downarrow\leftrightarrow$ <code>+search</code>	Argument defined by the cursor is used as search key.
<code>-search</code>	Search backwards (from the cursor towards the beginning of the file) for an occurrence of the search key last used.
<code>enter</code> <code>-search</code>	Search key used is from cursor position up to next blank.
<code>enter</code> <code>word</code> <code>-search</code>	Search for the next occurrence of <code>word</code> .
<code>enter</code> $\uparrow\downarrow\leftrightarrow$ <code>-search</code>	Argument defined by the cursor is used as search key.
<code>use</code>	Switch to the file previously used.
<code>enter</code> <code>use</code>	Edit file, taking its name from cursor position up to next blank.
<code>enter</code> <code>name</code> <code>use</code>	Make the current window look at file <code>name</code> . A linenummer and/or searchkey can be specified (as in the <i>med</i> command).
<code>enter</code> $\uparrow\downarrow\leftrightarrow$ <code>use</code>	Argument defined by the cursor is used as filename.
<code>window</code>	Make another window on the real screen, so that you now have two, or three, or more windows. A "default file" is used. It may be set to look at a file using <code>use</code> , and all other functions work within this little window. If the cursor is on the first or last column of your window the line separating the two windows goes horizontally on the line where the cursor is. The separating line goes vertically if the cursor is on the first or last line of your window. You may have two windows looking at the same file. In fact, it is rather neat, since changes made by editing in either window are reflected (at reasonable intervals) in the other window.
<code>enter</code> <code>window</code>	Delete last created window and return to the previous one.
<code>enter</code> <code>name</code> <code>window</code>	Create another window displaying file <code>name</code> .

MACRO

Typing

`[macro]` keystrokes `[macro]` key
stores the **keystrokes** sequence into the **key**. To avoid trouble, **key** can only be a printable character. From now on pressing the **key** is equivalent to typing the (long) **keystrokes** sequence. The sequence `[macro] [macro] key` restores the original function of the **key**.

Miscellaneous

What do the funny characters in the margins mean?

- | is normal.
- ; means this is past the end of the file. You may still type stuff - there just wasn't anything there before. Even if you type something on a line, the character won't go away until the line is rewritten by the editor - but the stuff is still there.
- < There is still more text to the left (may be blanks).
- > There is still more text to the right (not only blanks).

When editing a file for which you don't have write permission, the appropriate editor functions will be disabled.

What to do when disaster comes:

You are protected from loss of files by the insurance system of the MED editor. If you edit a file named *foo*, the old file *foo* is renamed *foo.bak* (the old *foo.bak* is deleted). If you do not like the results of your edit, the UNIX command:

```
mv foo.bak foo
```

restores the original file *foo*.

FILES

`/tmp/MED*` temporary workfile (PICK- and CLOSE-buffer)
`$$SAVE/MED*` saves state of editing session
`/usr/lib/med/default` default file
`*.bak` backup files

SEE ALSO

`termcap(5)`, `curses(3)`, `keycap(5)`

AUTHOR

Dittmar Krall, Wolfratshausen, Germany.
Inspired by the RAND Editor, Steve Zucker e.a., Santa Monica, California.

BUGS

Editor crashes can leave your terminal in a strange state, e.g. with disabled keyboard. Your system administrator should have a command to enable the keyboard. My panic solution is to switch terminal off/on in order to continue.

NAME

`mesg` - permit or deny messages

SYNOPSIS

`mesg [n] [y]`

DESCRIPTION

Mesg with argument *n* forbids messages via *write*(1) by revoking non-user write permission on the user's terminal. *Mesg* with argument *y* reinstates permission. All by itself, *mesg* reports the current state without changing it.

FILES

`/dev/tty*`
`/dev`

SEE ALSO

`write`(1)

DIAGNOSTICS

Exit status is 0 if messages are receivable, 1 if not, 2 on error.

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

1000000

NAME

`mkdir` - make a directory

SYNOPSIS

`mkdir` dirname ...

DESCRIPTION

Mkdir creates specified directories in mode 777, as modified by *umask*(2). Standard entries, '.', for the directory itself, and '..' for its parent, are made automatically.

Mkdir requires write permission in the parent directory.

SEE ALSO

`rm`(1)

DIAGNOSTICS

Mkdir returns exit code 0 if all directories were successfully made. Otherwise it prints a diagnostic and returns nonzero.

1945-46

1945-46

1945-46

1945-46

1945-46

1945-46

1945-46

1945-46

1945-46

1945-46

1945-46

1945-46

1945-46

1945-46

NAME

`mkstr` - create an error message file by massaging C source

SYNOPSIS

`mkstr` [-] messagefile prefix file ...

DESCRIPTION

Mkstr is used to create files of error messages. Its use can make programs with large numbers of error diagnostics much smaller, and reduce system overhead in running the program as the error messages do not have to be constantly swapped in and out.

Mkstr will process each of the specified *files*, placing a massaged version of the input file in a file whose name consists of the specified *prefix* and the original name. A typical usage of *mkstr* would be

```
mkstr pistrings xx *.c
```

This command would cause all the error messages from the C source files in the current directory to be placed in the file *pistrings* and processed copies of the source for these files to be placed in files whose names are prefixed with *xx*.

To process the error messages in the source to the message file *mkstr* keys on the string 'error(' in the input stream. Each time it occurs, the C string starting at the '(' is placed in the message file followed by a null character and a new-line character; the null character terminates the message so it can be easily used when retrieved, the new-line character makes it possible to sensibly *cat* the error message file to see its contents. The massaged copy of the input file then contains a *lseek* pointer into the file which can be used to retrieve the message, i.e.:

```
char  efilename[] = "/usr/lib/pi_strings";
int   efil = -1;

error(a1, a2, a3, a4)
{
    char buf[256];

    if (efil < 0) {
        efil = open(efilename, 0);
        if (efil < 0) {
oops:
            perror(efilename);
            exit(1);
        }
    }
    if (lseek(efil, (long) a1, 0) || read(efil, buf, 256) <= 0)
        goto oops;
    printf(buf, a2, a3, a4);
}
```

The optional - causes the error messages to be placed at the end of the specified message file for recompiling part of a large *mkstr* ed program.

MKSTR(1)

MUNIX

MKSTR(1)

SEE ALSO

lseek(2), xstr(1)

AUTHORS

William Joy and Charles Haley

NAME

`mm` - print out documents formatted with the MM macros

SYNOPSIS

`mm` [options] [files]

DESCRIPTION

Mm can be used to type out documents using *nroff*(1) and the MM text-formatting macro package. It has options to specify preprocessing by *tbl*(1) and/or *neqn*(1) and postprocessing by various terminal-oriented output filters. The proper pipelines and the required arguments and flags for *nroff*(1) and MM are generated, depending on the options selected.

Options for *mm* are given below. Any other arguments or flags (e.g., `-rC3`) are passed to *nroff*(1) or to MM, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, *mm* prints a list of its options.

- `-Tterm` Specifies the type of output terminal; for a list of recognized values for *term*, type `help term2`. If this option is *not* used, *mm* will use the value of the shell variable `$TERM` from the environment (see *profile*(5) and *environ*(7)) as the value of *term*, if `$TERM` is set; otherwise, *mm* will use 450 as the value of *term*. If several terminal types are specified, the last one takes precedence.
- `-12` Indicates that the document is to be produced in 12-pitch. May be used when `$TERM` is set to one of 300, 300s, 450, and 1620. (The pitch switch on the DASI 300 and 300s terminals must be manually set to 12 if this option is used.)
- `-c` Causes *mm* to invoke *col*(1); note that *col*(1) is invoked automatically by *mm* unless *term* is one of 300, 300s, 450, 37, 4000A, 382, 4014, tek, 1620, and X.
- `-e` Causes *mm* to invoke *neqn*(1).
- `-t` Causes *mm* to invoke *tbl*(1).
- `-E` Invokes the `-e` option of *nroff*(1).
- `-y` Causes *mm* to use the non-compacted version of the macros (see *mm*(7)).

As an example (assuming that the shell variable `$TERM` is set in the environment to 450), the two command lines below are equivalent:

```
mm -t -rC3 -12 ghh*
tbl ghh* | nroff -cm -T450-12 -h -rC3
```

Mm reads the standard input when `-` is specified instead of any file names. (Mentioning other files together with `-` leads to disaster.) This option allows *mm* to be used as a filter, e.g.:

```
cat dws | mm -
```

HINTS

1. *Mm* invokes *nroff*(1) with the `-h` flag. With this flag, *nroff*(1) assumes that the terminal has tabs set every 8 character positions.
2. Use the `-olist` option of *nroff*(1) to specify ranges of pages to be output. Note, however, that *mm*, if invoked with one or more of

the `-e`, `-t`, and `-` options, *together* with the `-olist` option of `nroff(1)` may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

3. If you use the `-s` option of `nroff(1)` (to stop between pages of output), use line-feed (rather than return or new-line) to restart the output. The `-s` option of `nroff(1)` does not work with the `-c` option of `mm`, or if `mm` automatically invokes `col(1)` (see `-c` option above).
4. If you lie to `mm` about the kind of terminal its output will be printed on, you'll get (often subtle) garbage; however, if you are redirecting output into a file, use the `-T37` option, and then use the appropriate terminal filter when you actually print that file.

SEE ALSO

`col(1)`, `env(1)`, `eqn(1)`, `greek(1)`, `mmt(1)`, `nroff(1)`, `tbl(1)`, `profile(5)`, `mm(7)`, `term(7)`.

MM—Memorandum Macros by D. W. Smith and J. R. Mashey.

Typing Documents with MM by D. W. Smith and E. M. Piskorik.

DIAGNOSTICS

"mm: no input file" if none of the arguments is a readable file and `mm` is not used as a filter.

NAME

mmchek - check usage of mm macros and eqn delimiters

SYNOPSIS

mmchek [files]

DESCRIPTION

Mmchek is a program for checking the contents of the named *files* for errors in the use of Memorandum Macros (see *mm(1)*) and some *eqn(1)* constructions. Appropriate messages are produced. The program skips all directories, and if no file name is given, standard input is read.

SEE ALSO

eqn(1), *mm(1)*, *mmt(1)*.

MM-Memorandum Macros by D. W. Smith and J. R. Mashey.

DIAGNOSTICS

Unreadable files cause the message "Cannot open *file-name*". The remaining output of the program is diagnostic of the source file.

BUGS

This is an experimental version of *mmchek*. *Mmchek* may be fully supported in the future.

NAME

wachk - check usage of mm macros and spin delimiters

SYNOPSIS

wachk [files]

DESCRIPTION

Wachk is a program for checking the contents of the named files for errors in the use of Memory Management Macros (see mm(1)) and some other constructs. Appropriate messages are produced. The program skips all directories and if no file name is given, standard input is read.

SEE ALSO

mm(1), mm(1), mm(1),
MM-Memory Management Macros by D. W. Smith and J. R. Mashey

DIAGNOSTICS

Unreadable files cause the message "Cannot open file name". The contents of the program is described in the source file.

BUGS

This is an experimental version of wachk. It may not be fully supported in the future.

NAME

`mmt`, `mvt` - typeset documents, view graphs, and slides

SYNOPSIS

`mmt` [options] [files]

`mvt` [options] [files]

DESCRIPTION

These two commands are very similar to `mm(1)`, except that they both typeset their input via `troff(1)`, as opposed to formatting it via `nroff(1)`; `mmt` uses the MM macro package, while `mvt` uses the Macro Package for View Graphs and Slides. These two commands have options to specify preprocessing by `tbl(1)` and/or `eqn(1)`. The proper pipelines and the required arguments and flags for `troff(1)` and for the macro packages are generated, depending on the options selected.

Options are given below. Any other arguments or flags (e.g., `-rC3`) are passed to `troff(1)` or to the macro package, as appropriate. Such options can occur in any order, but they must appear before the *files* arguments. If no arguments are given, these commands print a list of their options.

- `-e` Causes these commands to invoke `eqn(1)`.
- `-t` Causes these commands to invoke `tbl(1)`.
- `-Tst` Directs the output to the MH STARE facility.
- `-T4014` Directs the output to a Tektronix 4014 terminal via the `tc(1)` filter.
- `-Ttek` Same as `-T4014`.
- `-a` Invokes the `-a` option of `troff(1)`.
- `-y` Causes `mmt` to use the non-compacted version of the macros (see `mm(7)`). No effect for `mvt`.

These commands read the standard input when `-` is specified instead of any file names.

`Mvt` is just a link to `mmt`.

HINT

Use the `-olist` option of `troff(1)` to specify ranges of pages to be output. Note, however, that these commands, if invoked with one or more of the `-e`, `-t`, and `-` options, *together* with the `-olist` option of `troff(1)` may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

SEE ALSO

`env(1)`, `eqn(1)`, `mm(1)`, `tbl(1)`, `tc(1)`, `troff(1)`, `profile(5)`, `environ(7)`, `mm(7)`, `mv(7)`.

MM—Memorandum Macros by D. W. Smith and J. R. Mashey.

Typing Documents with MM by D. W. Smith and E. M. Piskorik.

A Macro Package for View Graphs and Slides by T. A. Dolotta and D. W. Smith (in preparation).

DIAGNOSTICS

"`m[mv]t: no input file`" if none of the arguments is a readable file and the command is not used as a filter.

...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...

...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...

...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...

...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...

...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...

...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...

...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...

...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...

...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...

...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...

NAME

more, page - file perusal filter for crt viewing

SYNOPSIS

```
more [ -cdfisu ] [ -n ] [ +linenumber ] [ +pattern ] [ name ... ]
page more options
```

DESCRIPTION

More is a filter which allows examination of a continuous text one screenful at a time on a soft-copy terminal. It normally pauses after each screenful, printing --More-- at the bottom of the screen. If the user then types a carriage return, one more line is displayed. If the user hits a space, another screenful is displayed. Other possibilities are enumerated later.

The command line options are:

- n* An integer which is the size (in lines) of the window which *more* will use instead of the default.
- c* *More* will draw each page by beginning at the top of the screen and erasing each line just before it draws on it. This avoids scrolling the screen, making it easier to read while *more* is writing. This option will be ignored if the terminal does not have the ability to clear to the end of a line.
- d* *More* will prompt the user with the message "Hit space to continue, Rubout to abort" at the end of each screenful. This is useful if *more* is being used as a filter in some setting, such as a class, where many users may be unsophisticated.
- f* This causes *more* to count logical, rather than screen lines. That is, long lines are not folded. This option is recommended if *nroff* output is being piped through *ul*, since the latter may generate escape sequences. These escape sequences contain characters which would ordinarily occupy screen positions, but which do not print when they are sent to the terminal as part of an escape sequence. Thus *more* may think that lines are longer than they actually are, and fold lines erroneously.
- l* Do not treat ^L (form feed) specially. If this option is not given, *more* will pause after any line that contains a ^L, as if the end of a screenful had been reached. Also, if a file begins with a form feed, the screen will be cleared before the file is printed.
- s* Squeeze multiple blank lines from the output, producing only one blank line. Especially helpful when viewing *nroff* output, this option maximizes the useful information present on the screen.
- u* Normally, *more* will handle underlining such as produced by *nroff* in a manner appropriate to the particular terminal: if the terminal can perform underlining or has a stand-out mode, *more* will output appropriate escape sequences to enable underlining or stand-out mode for underlined information in the source file. The *-u* option suppresses this processing.

+linenumber

Start up at *linenumber*.

+/*pattern*

Start up two lines before the line containing the regular expression *pattern*.

If the program is invoked as *page*, then the screen is cleared before each screenful is printed (but only if a full screenful is being printed), and $k - 1$ rather than $k - 2$ lines are printed in each screenful, where k is the number of lines the terminal can display.

More looks in the file */etc/termcap* to determine terminal characteristics, and to determine the default window size. On a terminal capable of displaying 24 lines, the default window size is 22 lines.

More looks in the environment variable *MORE* to pre-set any flags desired. For example, if you prefer to view files using the *-c* mode of operation, the *cs*h command *setenv MORE -c* or the *sh* command sequence *MORE='c'* ; *export MORE* would cause all invocations of *more*, including invocations by programs such as *man* and *msgs*, to use this mode. Normally, the user will place the command sequence which sets up the *MORE* environment variable in the *.cshrc* or *.profile* file.

If *more* is reading from a file, rather than a pipe, then a percentage is displayed along with the *--More--* prompt. This gives the fraction of the file (in characters, not lines) that has been read so far.

Other sequences which may be typed when *more* pauses, and their effects, are as follows (*i* is an optional integer argument, defaulting to 1) :

- i* <space> display *i* more lines, (or another screenful if no argument is given)
- ~D* display 11 more lines (a "scroll"). If *i* is given, then the scroll size is set to *i*.
- d* same as *~D* (control-D)
- iz* same as typing a space except that *i*, if present, becomes the new window size.
- is* skip *i* lines and print a screenful of lines
- if* skip *i* screenfuls and print a screenful of lines
- q* or *Q* Exit from *more*.
- =* Display the current line number.
- v* Start up the editor *vi* at the current line.
- h* Help command; give a description of all the *more* commands.
- i* /*expr* search for the *i*-th occurrence of the regular expression *expr*. If there are less than *i* occurrences of *expr*, and the input is a file (rather than a pipe), then the position in the file remains unchanged. Otherwise, a screenful is displayed, starting two lines before the place where the expression was found. The user's erase and kill characters may be used to edit the regular expression. Erasing back past the first column cancels the search command.
- in* search for the *i*-th occurrence of the last regular expression entered.

(single quote) Go to the point from which the last search started. If no search has been performed in the current file, this command goes back to the beginning of the file.

!command

invoke a shell with *command*. The characters '%' and '!' in "command" are replaced with the current file name and the previous shell command respectively. If there is no current file name, '%' is not expanded. The sequences "\%" and "\!" are replaced by "%" and "!" respectively.

i:n skip to the *i*-th next file given in the command line (skips to last file if *n* doesn't make sense)

i:p skip to the *i*-th previous file given in the command line. If this command is given in the middle of printing out a file, then *more* goes back to the beginning of the file. If *i* doesn't make sense, *more* skips back to the first file. If *more* is not reading from a file, the bell is rung and nothing else happens.

:f display the current file name and line number.

:q or :Q

exit from *more* (same as q or Q).

(dot) repeat the previous command.

The commands take effect immediately, i.e., it is not necessary to type a carriage return. Up to the time when the command character itself is given, the user may hit the line kill character to cancel the numerical argument being formed. In addition, the user may hit the erase character to redisplay the --More--(xx%) message.

At any time when output is being sent to the terminal, the user can hit the quit key (normally control- \backslash). *More* will stop sending output, and will display the usual --More-- prompt. The user may then enter one of the above commands in the normal manner. Unfortunately, some output is lost when this is done, due to the fact that any characters waiting in the terminal's output queue are flushed when the quit signal occurs.

The terminal is set to *noecho* mode by this program so that the output can be continuous. What you type will thus not show on your terminal, except for the / and ! commands.

If the standard output is not a teletype, then *more* acts just like *cat*, except that a header is printed before each file (if there is more than one).

A sample usage of *more* in previewing *nroff* output would be

```
nroff -ms +2 doc.n | more -s
```

AUTHOR

Eric Shienbrood, minor revisions by John Foderaro and Geoffrey Peck

FILES

```
/etc/termcap           Terminal data base
/usr/lib/more.help      Help file
```

SEE ALSO

csh(1), man(1), msgs(1), script(1), sh(1), environ(7)

CONFIDENTIAL - This document contains information which is exempt from public release under the provisions of the Freedom of Information Act, 5 U.S.C. 552, and is to be controlled and handled accordingly.

1. The purpose of this document is to provide information regarding the activities of the [redacted] and the [redacted] in the [redacted] area. This information is being provided for your information and is not to be disseminated outside of your office.

2. The information contained in this document is classified as [redacted] and is to be controlled and handled accordingly.

3. The information contained in this document is to be used for [redacted] purposes only and is not to be disseminated outside of your office. It is to be controlled and handled accordingly.

4. This document is to be controlled and handled accordingly.

5. The information contained in this document is to be controlled and handled accordingly.

6. The information contained in this document is to be controlled and handled accordingly.

7. The information contained in this document is to be controlled and handled accordingly.

8. The information contained in this document is to be controlled and handled accordingly.

9. The information contained in this document is to be controlled and handled accordingly.

10. This document is to be controlled and handled accordingly.

11. The information contained in this document is to be controlled and handled accordingly.

12. The information contained in this document is to be controlled and handled accordingly.

13. The information contained in this document is to be controlled and handled accordingly.

NAME

msgs - system messages and junk mail program

SYNOPSIS

msgs [-fhlpq] [number] [-number]

DESCRIPTION

Msgs is used to read system messages. These messages are sent by mailing to the login '*msgs*' and should be short pieces of information which are suitable to be read once by most users of the system.

Msgs is normally invoked each time you login, by placing it in the file *.login* (*.profile* if you use */bin/sh*). It will then prompt you with the source and subject of each new message. If there is no subject line, the first few non-blank lines of the message will be displayed. If there is more to the message, you will be told how long it is and asked whether you wish to see the rest of the message. The possible responses are:

y type the rest of the message

RETURN

synonym for y.

n skip this message and go on to the next message.

- redisplay the last message.

q drops you out of *msgs*; the next time you run the program it will pick up where you left off.

s append the current message to the file "Messages" in the current directory; 's-' will save the previously displayed message. A 's' or 's-' may be followed by a space and a filename to receive the message replacing the default "Messages".

m or 'm-' causes a copy of the specified message to be placed in a temporary mailbox and *mail*(1) to be invoked on that mailbox. Both 'm' and 's' accept a numeric argument in place of the '-'.

Msgs keeps track of the next message you will see by a number in the file *.msgsrc* in your home directory. In the directory */usr/msgs* it keeps a set of files whose names are the (sequential) numbers of the messages they represent. The file */usr/msgs/bounds* shows the low and high number of the messages in the directory so that *msgs* can quickly determine if there are no messages for you. If the contents of *bounds* is incorrect it can be fixed by removing it; *msgs* will make a new *bounds* file the next time it is run.

Options to *msgs* include:

-f which causes it not to say "No new messages.". This is useful in your *.login* file since this is often the case here.

-q Queries whether there are messages, printing "There are new messages." if there are. The command "*msgs -q*" is often used in login scripts.

-h causes *msgs* to print the first part of messages only.

-l option causes only locally originated messages to be reported.

num A message number can be given on the command line, causing *msgs* to start at the specified message rather than at the next message indicated by your *.msgsrc* file. Thus

msgs -h 1

prints the first part of all messages.

-number

will cause *msgs* to start *number* messages back from the one indicated by your *.msgsrc* file, useful for reviews of recent messages.

-p causes long messages to be piped through *more*(1).

Within *msgs* you can also go to any specific message by typing its number when *msgs* requests input as to what to do.

FILES

/usr/msgs/*
~/.msgsrc

database
number of next message to be presented

AUTHORS

William Joy
David Wasley

SEE ALSO

mail(1), more(1)

NAME

mt - magnetic tape manipulating program

SYNOPSIS

mt [-t *tapename*] *command* [*count*]

DESCRIPTION

Mt is used to give commands to the tape drive. If a tape name is not specified, /dev/nrmt0 is used. If a count is not specified, 1 is assumed.

Here are the commands:

eof	write <i>count</i> end-of-file marks
fsf	space forward <i>count</i> files
fsr	space forward <i>count</i> records
bsf	space backward <i>count</i> files
bsr	space backward <i>count</i> records
rew	rewind tape
offl	rewind tape and go offline
sw	swap bytes
nsw	do not swap bytes
len	read the tape; print size of each record

FILES

/dev/rmt* Raw magnetic tape interface

SEE ALSO

tm(4), dd(1)

1. The first part of the report is a general introduction to the subject.

2. The second part is a detailed description of the methods used.

3. The third part is a discussion of the results.

4. The fourth part is a conclusion and a summary of the findings.

5. The fifth part is a list of references.

6. The sixth part is a list of figures and tables.

7. The seventh part is a list of abbreviations.

8. The eighth part is a list of symbols.

9. The ninth part is a list of units.

10. The tenth part is a list of definitions.

11. The eleventh part is a list of footnotes.

12. The twelfth part is a list of appendices.

13. The thirteenth part is a list of indexes.

14. The fourteenth part is a list of errata.

15. The fifteenth part is a list of acknowledgments.

16. The sixteenth part is a list of references.

NAME

newgrp - log in to a new group

SYNOPSIS

newgrp [group]

DESCRIPTION

Newgrp changes the group identification of its caller, analogously to *login*(1). The same person remains logged in, and the current directory is unchanged, but calculations of access permissions to files are performed with respect to the new group ID.

Newgrp without an argument changes the group identification to the group in the password file; in effect it changes the group identification back to the caller's original group.

A password is demanded if the group has a password and the user himself does not, or if the group has a password and the user is not listed in */etc/group* as being a member of that group.

When most users log in, they are members of the group named *other*.

FILES

/etc/group
/etc/passwd

SEE ALSO

login(1), *group*(5).

BUGS

There is no convenient way to enter a password into */etc/group*. Use of group passwords is not encouraged, because, by their very nature, they encourage poor security practices. Group passwords may disappear in the future.

NAME

`news` - print news items

SYNOPSIS

`news [-a] [-n] [-s] [items]`

DESCRIPTION

News is used to keep the user informed of current events. By convention, these events are described by files in the directory `/usr/news`.

When invoked without arguments, *news* prints the contents of all current files in `/usr/news`, most recent first, with each preceded by an appropriate header. *News* stores the "currency" time as the modification date of a file named `.news_time` in the user's home directory (the identity of this directory is determined by the environment variable `$HOME`); only files more recent than this currency time are considered "current."

The `-a` option causes *news* to print all items, regardless of currency. In this case, the stored time is not changed.

The `-n` option causes *news* to report the names of the current items without printing their contents, and without changing the stored time.

The `-s` option causes *news* to report how many current items exist, without printing their names or contents, and without changing the stored time. It is useful to include such an invocation of *news* in one's `.profile` file, or in the system's `/etc/profile`.

All other arguments are assumed to be specific news items that are to be printed.

If a *delete* is typed during the printing of a news item, printing stops and the next item is started. Another *delete* within one second of the first causes the program to terminate.

FILES

`/etc/profile`
`/usr/news/*`
`$HOME/.news_time`

SEE ALSO

`profile(5)`, `environ(7)`.

REMARKS

The USENET news package is also very suitable for local news. It is much more powerful than this small news facility (see *inews(1)*, *postnews(1)*, *readnews(1)*).

1911

January 1st 1911

I have just received your letter of the 28th inst. and am glad to hear from you. I am well and hope this finds you the same. I have been thinking of you very much lately and wondering how you are getting on. I hope you are happy and contented. I have been very busy lately but I will try to write you more often.

I have been thinking of you very much lately and wondering how you are getting on. I hope you are happy and contented. I have been very busy lately but I will try to write you more often. I have been thinking of you very much lately and wondering how you are getting on. I hope you are happy and contented. I have been very busy lately but I will try to write you more often.

I have been thinking of you very much lately and wondering how you are getting on. I hope you are happy and contented. I have been very busy lately but I will try to write you more often. I have been thinking of you very much lately and wondering how you are getting on. I hope you are happy and contented. I have been very busy lately but I will try to write you more often.

I have been thinking of you very much lately and wondering how you are getting on. I hope you are happy and contented. I have been very busy lately but I will try to write you more often. I have been thinking of you very much lately and wondering how you are getting on. I hope you are happy and contented. I have been very busy lately but I will try to write you more often.

I have been thinking of you very much lately and wondering how you are getting on. I hope you are happy and contented. I have been very busy lately but I will try to write you more often. I have been thinking of you very much lately and wondering how you are getting on. I hope you are happy and contented. I have been very busy lately but I will try to write you more often.

Yours truly,

NAME

nice, *nohup* - run a command at low priority

SYNOPSIS

nice [*-number*] *command* [*arguments*]

nohup *command* [*arguments*]

DESCRIPTION

Nice executes *command* with low scheduling priority. If the *number* argument is present, the priority is incremented (higher numbers mean lower priorities) by that amount up to a limit of 20. The default *number* is 10.

The super-user may run commands with priority higher than normal by using a negative priority, e.g. '--10'.

Nohup executes *command* immune to hangup and terminate signals from the controlling terminal. The priority is incremented by 5. *Nohup* should be invoked from the shell with '&' in order to prevent it from responding to interrupts by or stealing the input from the next person who logs in on the same terminal.

FILES

nohup.out standard output and standard error file under *nohup*

SEE ALSO

nice(2)

DIAGNOSTICS

Nice returns the exit status of the subject command.

Copy sent to the [illegible] [illegible]

[illegible] [illegible] [illegible] [illegible]

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

NAME

`nl` - line numbering filter

SYNOPSIS

`nl [-htype] [-btype] [-ftype] [-vstart#] [-iincr] [-p] [-lnum] [-sssep]
[-wwidth] [-nformat] file`

DESCRIPTION

`Nl` reads lines from the named *file* or the standard input if no *file* is named and reproduces the lines on the standard output. Lines are numbered on the left in accordance with the command options in effect.

`Nl` views the text it reads in terms of logical pages. Line numbering is reset at the start of each logical page. A logical page consists of a header, a body, and a footer section. Empty sections are valid. Different line numbering options are independently available for header, body, and footer (e.g. no numbering of header and footer lines while numbering blank lines only in the body).

The start of logical page sections are signaled by input lines containing nothing but the following character(s):

Line contents	Start of
<code>\:\:\:</code>	header
<code>\:\:</code>	body
<code>\:</code>	footer

Unless signaled otherwise, `nl` assumes the text being read is in a single logical page body.

Command options may appear in any order and may be intermingled with an optional file name. Only one file may be named. The options are:

- `-btype` Specifies which logical page body lines are to be numbered. Recognized *types* and their meaning are: *a*, number all lines; *t*, number lines with printable text only; *n*, no line numbering; *pstring*, number only lines that contain the regular expression specified in *string*. Default *type* for logical page body is *t* (text lines numbered).
- `-htype` Same as `-btype` except for header. Default *type* for logical page header is *n* (no lines numbered).
- `-ftype` Same as `-btype` except for footer. Default for logical page footer is *n* (no lines numbered).
- `-p` Do not restart numbering at logical page delimiters.
- `-vstart#` *Start#* is the initial value used to number logical page lines. Default is 1.
- `-iincr` *Incr* is the increment value used to number logical page lines. Default is 1.
- `-sssep` *Sep* is the character(s) used in separating the line number and the corresponding text line. Default *sep* is a tab.
- `-wwidth` *Width* is the number of characters to be used for the line number. Default *width* is 6.

-nformat

Format is the line numbering format. Recognized values are: *ln*, left justified, leading zeroes suppressed; *rn*, right justified, leading zeroes suppressed; *rz*, right justified, leading zeroes kept. Default *format* is *rn* (right justified).

-lnum

Num is the number of blank lines to be considered as one. For example, *-l2* results in only the second adjacent blank being numbered (if the appropriate *-ha*, *-ba*, and/or *-fa* option is set). Default is 1.

SEE ALSO

pr(1).

NAME

nm - print name list

SYNOPSIS

nm [-gnopru] [file ...]

DESCRIPTION

Nm prints the name list (symbol table) of each object *file* in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced. If no *file* is given, the symbols in 'a.out' are listed.

Each symbol name is preceded by its value (blanks if undefined) and one of the letters U (undefined), A (absolute), T (text segment symbol), D (data segment symbol), B (bss segment symbol), or C (common symbol). If the symbol is local (non-external) the type letter is in lower case. The output is sorted alphabetically.

Options are:

- g Print only global (external) symbols.
- n Sort numerically rather than alphabetically.
- o Prepend file or archive element name to each output line rather than only once.
- p Don't sort; print in symbol-table order.
- r Sort in reverse order.
- u Print only undefined symbols.

SEE ALSO

ar(1), ar(5), a.out(5)

The first part of the report is a general description of the project and its objectives. It is followed by a detailed description of the methods used in the study.

The results of the study are presented in the following section. They show that the project has been successful in achieving its objectives. The data collected during the study are presented in the following tables.

The data collected during the study are presented in the following tables. The first table shows the results of the first experiment. The second table shows the results of the second experiment.

The data collected during the study are presented in the following tables. The third table shows the results of the third experiment. The fourth table shows the results of the fourth experiment.

The data collected during the study are presented in the following tables. The fifth table shows the results of the fifth experiment. The sixth table shows the results of the sixth experiment.

The data collected during the study are presented in the following tables. The seventh table shows the results of the seventh experiment. The eighth table shows the results of the eighth experiment.

The data collected during the study are presented in the following tables. The ninth table shows the results of the ninth experiment. The tenth table shows the results of the tenth experiment.

The data collected during the study are presented in the following tables. The eleventh table shows the results of the eleventh experiment. The twelfth table shows the results of the twelfth experiment.

NAME

notes, autoseq, nfpipe, nfprint, nfstats - a news system

SYNOPSIS

```
notes [ -sxin ] [ -t termtype ] topic1 [ -f file ] [ ... ]
autoseq
nfpipe topic [ -t title ] [ -d ] [ -a ]
nfprint [ -nn ] topic [ note list ]
nfstats [ -s ] topic [ ... ]
checknotes [ -qnvs ]
```

DESCRIPTION

Notes supports computer managed discussion forums. It coordinates access to and updates of data bases of notes and their responses. A single *notesfile* contains an ordered list of *base notes*, each of which may have an ordered list of responses associated with it. A *note string* consists of a *base note* and all of its responses. Separate *notesfiles* contain discussions on separate subject matters; microcomputers might be discussed in a "micronotes" *notesfile* while bicycling enthusiasts make their comments in a "bicycle" *notesfile*.

The **-s** option signals *notes* to use the automatic sequencer. With the sequencer enabled, *notes* shows the new notes and responses since your last entry into that *notesfile*. With the sequencer enabled by **-s** the notes program will not enter *notesfiles* which have no new text. Specify **-x** to use the sequencer and enter *notesfiles* even if they have no new text. The **-i** and **-n** options are still more sequencing modes, **-i** is similar to **-s** but shows the index page instead of the first modified note. **-n** turns the sequencer off.

Specify **-t termtype** to override the TERM environment variable. This switch is mostly used on version 6 systems.

The **-f** option directs *notes* to read the contents of a file for a list of *notesfiles* to scan.

The **topic list** specifies which *notesfiles* are to be scanned. The *notesfiles* are scanned from left to right; upon finishing the first topic, the second is entered. The shell's meta-characters are recognized within a topic but must be escaped to prevent shell interpretation. Specifying "net.*" will yield all the *notesfiles* with the prefix "net.". Specify "*unix*" to read all *notesfiles* with the string "unix" in their names. Bracket and question mark constructs are also recognized.

The *autoseq* feature allows sequencing through a list of *notesfiles* with the sequencer enabled using with a single command. The *autoseq* is (almost) syntactically equivalent to "notes -s \$NFSEQ" where NFSEQ contains a comma separated list of *notesfile* names. A typical NFSEQ definition for the Bourne shell looks like:

```
NFSEQ="general,announce,net.*,bicycle,srg".
```

Notes and responses are entered by using an editor. The default editor is */bin/ed*. This can be changed by setting one of the environment variables *NFED* or *EDITOR*.

Some commonly used commands within the notesfile system are listed below:

- space Show the next page of the note/response.
- ;
Go the next response, if there are no more responses go to the next note.
- Go to the previous page of the current note/response. From the first page of a response, go to the previous response (or the base note from the first response). From the first page of a base note, go to the previous note.
- newline Go to the next note.
- j Jump to the next unread note/response (when using sequencer).
- J Jump to the next unread note, ignoring any further responses in the current note string (when using sequencer).
- w When issued from the index page enters a new note. When entered from a note/response display enters a response. A capital-W will include the text of the currently displayed note/response in the new response.
- q Leave the current notesfile.
- Q Leave the current notesfile without updating the sequencer information.
- control-z Return to the shell, ignoring any further notesfiles in the current invocation. No sequencer information is updated.
- x Search for a note with the (prompted for) string in its title. Capital-X asks for a new search string, otherwise the last entered string is used.
- s Saves the currently displayed note/response at the end of a (prompted for) file. Capital-S saves the entire note string.
- M Sends the text of the note/response displayed and your comments to another user(s). The P command routes the letter to the author of the note/response.
- t Issues a write(1) command to the author of the currently displayed note/response. No action is taken if the note originated on a remote system or is anonymous.
- ! Forks a shell.

Nfpipe reads standard input to create a note in the specified notesfile. The title of the note is specified using the **-t** parameter. The **-d** flag specifies that this note is to have the director message flag enabled; if the author has no director privileges in the notesfile, this flag has no effect. Specifying **-a** makes the note anonymous; if anonymous notes are not permitted this flag has no effect.

Nfprint gives the user the ability to print the contents of notesfiles. The **-nn** parameter specifies the page length to use (66 lines/page is the default). The **note list** is the set of notes which are to be printed. An

example note list is: 1,30-36,13,10,42-50. **Nfprint** writes to standard output.

Nfstats prints usage statistics for notesfiles. This program takes a list of notesfiles and lists their statistics on standard output. If more than one notesfile is specified, an aggregate set of statistics is also produced. Specify **-s** to have only this aggregate report printed and the individual reports suppressed.

Checknotes reports on the existence of new notes. The **NFSEQ** environment variable is read to determine the users subscription. **Checknotes** then looks at each of if there are new notes. The various flags specify the method of notification. Use **-q** to have the message "There are new notes" printed if this is so. Specify **-n** to get a message "There are no new notes" if this is the case. With the **-v** option enabled, **checknotes** prints the names of the notesfiles which have new notes in them. The **-s** option is silent; no output is produced. Regardless of the option, the program exits with 0 (TRUE) if new notes exist and with 1 (FALSE) if no new notes exist.

An interface is provided to **news(1)**. Transfers in both directions are supported. See **notes(8)** for more information on how this works.

Only the **notesfile owner** can create new notesfiles. The **notesfile owner** will create the notesfile and turn control over to the person requesting the notesfile. This person is the **notesfile director**; he may designate others to also be **notesfile directors**. The **notesfile director** has special privileges including: deleting any note, determining policy for the notesfile, permitting anonymous notes, and determining accessibility of the notesfile.

The concept of a notesfile was taken from the PLATO system (a trademark of Control Data Corporation) designed at the University of Illinois to provide automated teaching capabilities.

BUGS

Only the name of a note's originating system is kept; another utility must be used to generate paths for mailing to distant (more than 1 hop) authors.

Some of the notesfile programs do not check to make sure the user is not the ANONUID user who is not allowed to run *notes*; the programs that fail to do this are ones which do not allow writing.

Sometimes interrupting a display with the RUBOUT or DELETE key will leave the program in a state where it prints only the first character of titles and other headers.

When more then 128 lines are saved with the "S" or "s" options of notes, the program reports strange values. This is because only 8 bits are available for passing the result.

FILES

/etc/passwd	for the users name
/etc/group	for the users group
/etc/termcap	for terminal capabilities
/usr/spool/notes	the notesfile data base

/usr/spool/notes/.utilities utility programs and online help
/usr/spool/notes/topic/text text of the notes in notesfile 'topic'
/usr/spool/notes/topic/note.indx the note descriptors for notesfile 'topic'
/usr/spool/notes/topic/resp.indx the response descriptors for notesfile 'topic'
/usr/spool/notes/topic/access the permission list for the notesfile 'topic'

SEE ALSO

ed(1), news(1), nfcomment(3), notes(8), termcap(3), write(1),
The Notesfile Reference Manual

AUTHORS

Ray Essick (uiucdcs!essick, uiucdcs!notes)
Rob Kolstad (uiucdcs!kolstad)
Department of Computer Science
222 Digital Computer Laboratory
University of Illinois at Urbana-Champaign
1304 West Springfield Ave.
Urbana, IL 61801

NAME

od - octal dump

SYNOPSIS

od [-bcdox] [file] [[+]offset[.][b]]

DESCRIPTION

Od dumps *file* in one or more formats as selected by the first argument. If the first argument is missing, -o is default. The meanings of the format argument characters are:

- b Interpret bytes in octal.
- c Interpret bytes in ASCII. Certain non-graphic characters appear as C escapes: null=\0, backspace=\b, formfeed=\f, newline=\n, return=\r, tab=\t; others appear as 3-digit octal numbers.
- d Interpret words in decimal.
- o Interpret words in octal.
- x Interpret words in hex.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The offset argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as octal bytes. If '.' is appended, the offset is interpreted in decimal. If 'b' is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded '+'.
Dumping continues until end-of-file.

SEE ALSO

adb(1), xd(1)

NAME

pack - packs or unpacks many files <---> one.

SYNOPSIS

- a) **pack** file1 [file2] ... [filen] or
- b) **pack**

DESCRIPTION

- a) **pack** file1 ... filen
packs these files onto stdout. Between the files the following line is inserted
{/*****<file i>*****/}
- b) **pack**
unpacks stdin (a previously packed file) onto the component files.
Hint: `fgrep '^{' packedfile`
picks out the names of the component files).

EXAMPLE1

pack *.p | xref
makes a cross-reference list of a group of files.

EXAMPLE2

pack *.c | sed -f change.sed | pack
seds a whole group of files at once.

DIAGNOSTICS

If file cannot be opened the message
'cannot open file <file>' is written on stderr.

If the inputfile has a bad format the message
'phase error in line <line number>' is written on stderr.

1. The first part of the document is a list of the names of the persons who were present at the meeting.

2. The second part of the document is a list of the names of the persons who were absent from the meeting.

3. The third part of the document is a list of the names of the persons who were present at the meeting.

4. The fourth part of the document is a list of the names of the persons who were absent from the meeting.

5. The fifth part of the document is a list of the names of the persons who were present at the meeting.

6. The sixth part of the document is a list of the names of the persons who were absent from the meeting.

7. The seventh part of the document is a list of the names of the persons who were present at the meeting.

8. The eighth part of the document is a list of the names of the persons who were absent from the meeting.

9. The ninth part of the document is a list of the names of the persons who were present at the meeting.

10. The tenth part of the document is a list of the names of the persons who were absent from the meeting.

11. The eleventh part of the document is a list of the names of the persons who were present at the meeting.

12. The twelfth part of the document is a list of the names of the persons who were absent from the meeting.

13. The thirteenth part of the document is a list of the names of the persons who were present at the meeting.

14. The fourteenth part of the document is a list of the names of the persons who were absent from the meeting.

15. The fifteenth part of the document is a list of the names of the persons who were present at the meeting.

16. The sixteenth part of the document is a list of the names of the persons who were absent from the meeting.

NAME

passwd - change login password

SYNOPSIS

passwd [name]

DESCRIPTION

This command changes (or installs) a password associated with the user *name* (your own name by default).

The program prompts for the old password and then for the new one. The caller must supply both. The new password must be typed twice, to forestall mistakes.

New passwords must be at least four characters long if they use a sufficiently rich alphabet and at least six characters long if monospace. These rules are relaxed if you are insistent enough.

Only the owner of the name or the super-user may change a password; the owner must prove he knows the old password.

FILES

/etc/passwd

SEE ALSO

login(1), passwd(5), crypt(3C)

Robert Morris and Ken Thompson, *Password Security: A Case History*

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

10-10-50

NAME

`paste` - merge same lines of several files or subsequent lines of one file

SYNOPSIS

```
paste file1 file2 ...
paste -dlist file1 file2 ...
paste -s [-dlist] file1 file2 ...
```

DESCRIPTION

In the first two forms, *paste* concatenates corresponding lines of the given input files *file1*, *file2*, etc. It treats each file as a column or columns of a table and pastes them together horizontally (parallel merging). If you will, it is the counterpart of *cat*(1) which concatenates vertically, i.e., one file after the other. In the last form above, *paste* subsumes the function of an older command with the same name by combining subsequent lines of the input file (serial merging). In all cases, lines are glued together with the *tab* character, or with characters from an optionally specified *list*. Output is to the standard output, so it can be used as the start of a pipe, or as a filter, if `-` is used in place of a file name.

The meanings of the options are:

- `-d` Without this option, the new-line characters of each but the last file (or last line in case of the `-s` option) are replaced by a *tab* character. This option allows replacing the *tab* character by one or more alternate characters (see below).
- list* One or more characters immediately following `-d` replace the default *tab* as the line concatenation character. The list is used circularly, i. e. when exhausted, it is reused. In parallel merging (i. e. no `-s` option), the lines from the last file are always terminated with a new-line character, not from the *list*. The list may contain the special escape sequences: `\n` (new-line), `\t` (tab), `\\` (backslash), and `\0` (empty string, not a null character). Quoting may be necessary, if characters have special meaning to the shell (e.g. to get one backslash, use `-d"\\\\"`).
- `-s` Merge subsequent lines rather than one from each input file. Use *tab* for concatenation, unless a *list* is specified with `-d` option. Regardless of the *list*, the very last character of the file is forced to be a new-line.
- `-` May be used in place of any file name, to read a line from the standard input. (There is no prompting).

EXAMPLES

```
ls | paste -d" " -      list directory in one column
ls | paste - - - -      list directory in four columns
paste -s -d"\t\n" file  combine pairs of lines into lines
```

SEE ALSO

`grep`(1), `cut`(1),
`pr`(1): `pr -t -m...` works similarly, but creates extra blanks, tabs and new-lines for a nice page layout.

DIAGNOSTICS

line too long

Output lines are restricted to 511 characters.

*too many files*Except for **-s** option, no more than 12 input files may be specified.

NAME

`pc` - Pascal compiler

SYNOPSIS

`pc [option] ... file ...`

DESCRIPTION

`pc` is the MUNIX Pascal compiler. It accepts several types of arguments:

Arguments whose names end with `.p` are taken to be Pascal source programs; they are compiled, and each object program is left on a file with suffix `.o`. The `.o` file is normally deleted, however, if a single Pascal program is compiled and loaded all at one go. Errors detected by the compiler are listed on *stdout*. This Pascal compiler has the *c++*-identifier `m68000` predefined, i.e. `"#ifdef m68000"` is true.

Filenames are built from the basename of the source program and a certain suffix.

A call `"pc test.p"` is the same as `"pc -c test.p && ld /lib/prt0.o test.o -lffp -lp -lc && rm test.o"`. The call `"pc *.o"` is the same as `"ld /lib/prt0.o *.o -lffp -lp -lc"`. If your directory contains the file `xyz.p`, a call `"make xyz"` will result in `"pc -o xyz xyz.p"`.

The following options are interpreted by `pc`. See *ld(1)* for load options.

- `-c` Suppress the loading phase of the compilation, and force an object file to be produced even if only one program is compiled.
- `-d` Switch on the debug mode.
- `-e` Display extension warning messages to *stdout*.
- `-m` Load **new, mark & release** instead of **new & dispose**.
- `-n` Suppress execution ("dry run") of `pc` commands.
- `-o output`
Name the final output file *output*. If this option is used the file `'a.out'` will be left undisturbed. This option is passed on to the linker.
- `-p` Arrange for the compiler to produce code which counts the number of times each routine is called; also, if loading takes place, replace the standard startup routine by one which automatically calls *monitor*(3C) at the start and arranges to write out a *mon.out* file at normal termination of execution of the object program. An execution profile can then be generated by use of *prof*(1).
- `-t[12]` Find only the designated compiler passes in the files whose names are constructed by a `-B` option. In the absence of a `-B` option, the *string* is taken to be `'/usr/src/cmd/pc/o'`.
- `-w` Display warning messages to *stdout*.
- `-Bstring`
Find substitute compiler passes in the files named *string* with the suffixes *pass1* and *pass2*. If *string* is empty, use a standard backup version.

-Dname=def

-Dname

Define the *name* to the preprocessor, as if by '#define'. If no definition is given, the name is defined as 1.

-Idir Files of '#include' type whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in -I options, then in directories on a standard list.

-L Additionally generates listings on corresponding files suffixed '.l'.

-P Run only the macro preprocessor and place the result for each '.p' file in a corresponding '.i' file and has no '#' lines in it.

-S Compile the named Pascal programs and leave the assembler-language output on corresponding files suffixed '.s'.

-T Trace and print *pc* commands. Temporary files are not deleted.

-Uname

Remove any initial definition of *name*.

Other arguments are taken to be either loader(ld) option arguments, or Pascal-compatible object programs, typically produced by an earlier *pc* run, or perhaps libraries of Pascal-compatible routines. These programs, together with the results of any compilations specified, are loaded (in the order given) to produce an executable program with name *a.out*.

FILES

file.i	preprocessor output file
file.l	error and listing file
file.o	object file
file.p	input file
file.s	assembler listing file
a.out	loaded (linked) output
/tmp/ptm*	temporary files for <i>pc</i>
/lib/cpp	preprocessor
/lib/pass[12]	compiler passes for <i>pc</i>
/lib/c2	compiler pass3
/lib/perror	prints errors and listing
/lib/prt0.o	runtime startoff
/lib/mprt0.o	startoff for profiling
/lib/libmark.a	pascal runtime-support (new, mark, release)
/lib/libffp.a	library with floating point routines
/lib/libp.a	pascal runtime-support
/lib/libc.a	standard library, see <i>intro</i> (3)
/usr/include/pc/init.h	initialisation of Pascal compilation
/usr/src/cmd/pc/opass[12]	backup compiler passes for <i>pc</i>

SEE ALSO

K. Jensen and N. Wirth *Pascal User Manual and Report* Springer Verlag
1978

monitor(3C), prof(1), adb(1), ld(1)

DIAGNOSTICS

The diagnostics produced by *pc* itself are intended to be self-explanatory. Occasional messages may be produced by the loader(ld).

BUGS

None known.

NAME

pg, pn — simple paginators for text files

SYNOPSIS

pg file

pn file

(any command) | pg

DESCRIPTION

pg and **pn** are very simple and small programs for displaying text files on a terminal. They can be called with one or more filenames or at the end of a pipe (for example **ll | pg**). One cannot alter the files being displayed with **pg**.

The only difference between **pg** and **pn** is that **pg** erases the terminal screen before and after paginating the text whereas **pn** does not.

pg displays files on a normal (24-line) terminal 23 lines at a time (wrapping lines longer than the screen width around) and waits after each screenfull for the user to type in one of the command characters listed below.

When displaying texts, the last line of the terminal is a prompt for user input. The prompt tells the user if there is more of the text file to come or if he is at the end.

pg has commands to look at different parts of a file. They are:

Command	Action
f	(Forwards) pg shows the next 23 lines of the file (if there are that many).
<RETURN>	same as f .
<SPACE>	same as f .
;	same as f .
d	(Down) pg scrolls the next 15 lines.
b	(Back) pg goes back to line 1 of the file. This does not work with pipes.
q	(Quit) switch to the next file (if there are more) or exit pg if not.
~C	exit pg
e	(Edit) pg calls (with exelcp) the user's environment editor (shell parameter EDITOR) on the text file being paginated.
?	Display help file for vsh display mode and pg .

pg is often better than **cat** because it automatically waits after each full terminal screen. Some people also prefer **pg** to **more** because one can go backwards in a file (ie. with '**b**').

BUGS

pg has an internal function that tries to make sure that a file to be paginated is indeed ascii text. It sometimes gets confused and lets one look at a core dump or directory. The command 'b' is inoperable on pipes because one can't rewind them... Otherwise **pg** has no known bugs.

AUTHOR

Stephen T. Pope at PCS GmbH, Munich, Germany

SEE ALSO

pg and **pn** are essentially the same as the internal paginator of **vsh** (1) and/or the **help** (1) utility.

NAME

plot - graphics filters

SYNOPSIS

plot [-Tterminal [raster]]

DESCRIPTION

These commands read plotting instructions (see *plot(5)*) from the standard input, and in general produce plotting instructions suitable for a particular *terminal* on the standard output.

If no *terminal* type is specified, the environment parameter \$TERM (see *environ(7)*) is used. Known *terminals* are:

4014 Tektronix 4014 storage scope.

450 DASI Hyterm 450 terminal (Diablo mechanism).

300 DASI 300 or GSI terminal (Diablo mechanism).

300S DASI 300S terminal (Diablo mechanism).

ver Versatec D1200A printer-plotter. This version of *plot* places a scan-converted image in '/usr/tmp/raster' and sends the result directly to the plotter device rather than to the standard output. The optional argument causes a previously scan-converted file *raster* to be sent to the plotter.

lbp Canon LBP10 laser beam printer.

bip CADMUS bit map display.

FILES

/usr/bin/tek
/usr/bin/t450
/usr/bin/t300
/usr/bin/t300s
/usr/bin/vplot
/usr/tmp/raster
/usr/bin/lplot
/usr/bin/bplot

SEE ALSO

plot(3X), plot(5)

BUGS

There is no lockout protection for /usr/tmp/raster.

NAME

postnews - submit news articles

SYNOPSIS

postnews [*article*]

DESCRIPTION

Postnews is a shell script that calls *inews*(1) to submit news articles to USENET. It should be edited to reflect your organizations name. It will prompt the user for the title of the article (which should be a phrase suggesting the subject, so that persons reading the news can tell if they are interested in the article) for the newsgroup, and for the distribution.

An omitted newsgroup (from hitting return) will default to *general*.

general is read by everyone on the local machine. Other possible newsgroups include, but are not limited to, *btl.general*, which is read by all users at all Bell Labs sites on USENET, *net.general*, which is read by all users at all sites on USENET, and *net.news*, which is read by users interested in the network news on all sites. There is often a local set of newsgroups, such as *ucb.all*, that circulate within a local set of machines. (In this case, *ucb* newsgroups circulate among machines at the University of California at Berkeley.)

The distribution can be any valid newsgroup name list, and defaults to the same as the newsgroup. (If they are the same, the distribution will be omitted from the headers put into the editor buffer.) A distribution header will, if given, be included in the headers of the article, affecting where the article is distributed to.

After entering the title, newsgroup, and distribution, the user will be placed in an editor. If *\$EDITOR* is set in the environment, that editor will be used. Otherwise, *postnews* defaults to *med*(1).

An initial set of headers containing the subject and newsgroups will be placed in the editor, followed by a blank line. The article should be appended to the buffer, after the blank line. These headers can be changed, or additional headers added, while in the editor, if desired.

Optionally, the article will be read from the specified *filename*.

For more sophisticated uses, such as posting news from a program, see *inews*(1).

FILES

SEE ALSO

checknews(1), inews(1), mail(1), readnews(1).

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

NAME

`pr` - print files

SYNOPSIS

`pr` [options] [files]

DESCRIPTION

Pr prints the named files on the standard output. If *file* is `-`, or if no files are specified, the standard input is assumed. By default, the listing is separated into pages, each headed by the page number, a date and time, and the name of the file.

By default, columns are of equal width, separated by at least one space; lines which do not fit are truncated. If the `-s` option is used, lines are not truncated and columns are separated by the separation character.

If the standard output is associated with a terminal, error messages are withheld until *pr* has completed printing.

Options may appear singly or be combined in any order. Their meanings are:

- `+k` Begin printing with page *k* (default is 1).
- `-k` Produce *k*-column output (default is 1). The options `-e` and `-i` are assumed for multi-column output.
- `-a` Print multi-column output across the page.
- `-m` Merge and print all files simultaneously, one per column (overrides the `-k`, and `-a` options).
- `-d` Double-space the output.
- `-n` Number the lines.
- `-eck` Expand *input* tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. Tab characters in the input are expanded into the appropriate number of spaces. If *c* (any non-digit character) is given, it is treated as the input tab character (default for *c* is the tab character).
- `-ick` In *output*, replace white space wherever possible by inserting tabs to character positions *k*+1, 2**k*+1, 3**k*+1, etc. If *k* is 0 or is omitted, default tab settings at every eighth position are assumed. If *c* (any non-digit character) is given, it is treated as the output tab character (default for *c* is the tab character).
- `-nck` Provide *k*-digit line numbering (default for *k* is 5). The number occupies the first *k*+1 character positions of each column of normal output or each line of `-m` output. If *c* (any non-digit character) is given, it is appended to the line number to separate it from whatever follows (default for *c* is a tab).
- `-wk` Set the width of a line to *k* character positions (default is 72 for equal-width multi-column output, no limit otherwise).
- `-ok` Offset each line by *k* character positions (default is 0). The number of character positions per line is the sum of the width and offset.

100

100

100

100

100

100

100

100

100

100

100

100

100

- lk** Set the length of a page to *k* lines (default is 66).
- h** Use the next argument as the header to be printed instead of the file name.
- p** Pause before beginning each page if the output is directed to a terminal (*pr* will ring the bell at the terminal and wait for a carriage return).
- f** Use form-feed character for new pages (default is to use a sequence of line-feeds). Pause before beginning the first page if the standard output is associated with a terminal.
- r** Print no diagnostic reports on failure to open files.
- t** Print neither the five-line identifying header nor the five-line trailer normally supplied for each page. Quit printing after the last line of each file without spacing to the end of the page.
- sc** Separate columns by the single character *c* instead of by the appropriate number of spaces (default for *c* is a tab).

EXAMPLES

Print **file1** and **file2** as a double-spaced, three-column listing headed by "file list":

```
pr -3dh "file list" file1 file2
```

Write **file1** on **file2**, expanding tabs to columns 10, 19, 28, 37, ...:

```
pr -e9 -t <file1 >file2
```

FILES

/dev/tty* to suspend messages

SEE ALSO

cat(1).

NAME

prep - prepare text for statistical processing

SYNOPSIS

prep [-dio] file ...

DESCRIPTION

Prep reads each *file* in sequence and writes it on the standard output, one 'word' to a line. A word is a string of alphabetic characters and imbedded apostrophes, delimited by space or punctuation. Hyphenated words are broken apart; hyphens at the end of lines are removed and the hyphenated parts are joined. Strings of digits are discarded.

The following option letters may appear in any order:

- d Print the word number (in the input stream) with each word.
- i Take the next *file* as an 'ignore' file. These words will not appear in the output. (They will be counted, for purposes of the -d count.)
- o Take the next *file* as an 'only' file. Only these words will appear in the output. (All other words will also be counted for the -d count.)
- p Include punctuation marks (single nonalphanumeric characters) as separate output lines. The punctuation marks are not counted for the -d count.

Ignore and only files contain words, one per line.

SEE ALSO

deroff(1)

NAME

prmail - print out mail in the post office

SYNOPSIS

prmail [user ...]

DESCRIPTION

Prmail prints the mail which waits for you, or the specified user, in the post office. The mail is not disturbed.

FILES

/usr/mail/* post office

SEE ALSO

mail(1), from(1)

NAME

prof - display profile data

SYNOPSIS

prof [*-v*] [*-a*] [*-l*] [*-low* [*-high*]] [file]

DESCRIPTION

Prof interprets the file *mon.out* produced by the *monitor* subroutine. Under default modes, the symbol table in the named object file (*a.out* default) is read and correlated with the *mon.out* profile file. For each external symbol, the percentage of time spent executing between that symbol and the next is printed (in decreasing order), together with the number of times that routine was called and the number of milliseconds per call.

If the *-a* option is used, all symbols are reported rather than just external symbols. If the *-l* option is used, the output is listed by symbol value rather than decreasing percentage.

If the *-v* option is used, all printing is suppressed and a graphic version of the profile is produced on the standard output for display by the *plot(1G)* filters. The numbers *low* and *high*, by default 0 and 100, cause a selected percentage of the profile to be plotted with accordingly higher resolution.

In order for the number of calls to a routine to be tallied, the *-p* option of *cc* must have been given when the file containing the routine was compiled. This option also arranges for the *mon.out* file to be produced automatically.

FILES

mon.out for profile
a.out for namelist

SEE ALSO

monitor(3C), *profil(2)*, *cc(1)*, *plot(1G)*

BUGS

Beware of quantization errors.

...

...

...

...

...

...

...

...

...

...

...

...

...

...

NAME

`prs` - print an SCCS file

SYNOPSIS

`prs [-d[dataspec]] [-r[SID]] [-e] [-l] [-a] files`

DESCRIPTION

Prs prints, on the standard output, parts or all of an SCCS file (see *sccsfile(5)*) in a user supplied format. If a directory is named, *prs* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with *s*.), and unreadable files are silently ignored. If a name of *-* is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file or directory to be processed; non-SCCS files and unreadable files are silently ignored.

Arguments to *prs*, which may appear in any order, consist of *keyletter* arguments, and file names.

All the described *keyletter* arguments apply independently to each named file:

- `-d[dataspec]` Used to specify the output data specification. The *dataspec* is a string consisting of SCCS file *data keywords* (see *DATA KEYWORDS*) interspersed with optional user supplied text.
- `-r[SID]` Used to specify the *SCCS IDentification* (SID) string of a delta for which information is desired. If no SID is specified, the SID of the most recently created delta is assumed.
- `-e` Requests information for all deltas created *earlier* than and including the delta designated via the *-r* keyletter.
- `-l` Requests information for all deltas created *later* than and including the delta designated via the *-r* keyletter.
- `-a` Requests printing of information for both removed, i.e., delta type = *R*, (see *rmDEL(1)*) and existing, i.e., delta type = *D*, deltas. If the *-a* keyletter is not specified, information for existing deltas only is provided.

DATA KEYWORDS

Data keywords specify which parts of an SCCS file are to be retrieved and output. All parts of an SCCS file (see *sccsfile(5)*) have an associated data keyword. There is no limit on the number of times a data keyword may appear in a *dataspec*.

The information printed by *prs* consists of: (1) the user supplied text; and (2) appropriate values (extracted from the SCCS file) substituted for the recognized data keywords in the order of appearance in the *dataspec*. The format of a data keyword value is either *Simple* (S), in which keyword substitution is direct, or *Multi-line* (M), in which keyword substitution is followed by a carriage return.

User supplied text is any text other than recognized data keywords. A tab is specified by \t and carriage return/new-line is specified by \n.

TABLE 1. SCCS Files Data Keywords

Keyword	Data Item	File Section	Value	Format
:Dt:	Delta information	Delta Table	See below*	S
:DL:	Delta line statistics	"	:Li:/:Ld:/:Lu:	S
:Li:	Lines inserted by Delta	"	nnnnn	S
:Ld:	Lines deleted by Delta	"	nnnnn	S
:Lu:	Lines unchanged by Delta	"	nnnnn	S
:DT:	Delta type	"	D or R	S
:I:	SCCS ID string (SID)	"	:R:/:L:/:B:/:S:	S
:R:	Release number	"	nnnn	S
:L:	Level number	"	nnnn	S
:B:	Branch number	"	nnnn	S
:S:	Sequence number	"	nnnn	S
:D:	Date Delta created	"	:Dy:/:Dm:/:Dd:	S
:Dy:	Year Delta created	"	nn	S
:Dm:	Month Delta created	"	nn	S
:Dd:	Day Delta created	"	nn	S
:T:	Time Delta created	"	:Th:/:Tm:/:Ts:	S
:Th:	Hour Delta created	"	nn	S
:Tm:	Minutes Delta created	"	nn	S
:Ts:	Seconds Delta created	"	nn	S
:P:	Programmer who created Delta	"	logname	S
:DS:	Delta sequence number	"	nnnn	S
:DP:	Predecessor Delta seq-no.	"	nnnn	S
:DI:	Seq-no. of deltas incl., excl., ignored	"	:Dn:/:Dx:/:Dg:	S
:Dn:	Deltas included (seq #)	"	:DS: :DS:...	S
:Dx:	Deltas excluded (seq #)	"	:DS: :DS:...	S
:Dg:	Deltas ignored (seq #)	"	:DS: :DS:...	S
:MR:	MR numbers for delta	"	text	M
:C:	Comments for delta	"	text	M
:UN:	User names	User Names	text	M
:FL:	Flag list	Flags	text	M
:Y:	Module type flag	"	text	S
:MF:	MR validation flag	"	yes or no	S
:MP:	MR validation pgm name	"	text	S
:KF:	Keyword error/warning flag	"	yes or no	S
:BF:	Branch flag	"	yes or no	S
:J:	Joint edit flag	"	yes or no	S
:LK:	Locked releases	"	:R:...	S
:Q:	User defined keyword	"	text	S
:M:	Module name	"	text	S
:FB:	Floor boundary	"	:R:	S
:CB:	Ceiling boundary	"	:R:	S
:Ds:	Default SID	"	:I:	S
:ND:	Null delta flag	"	yes or no	S
:FD:	File descriptive text	Comments	text	M
:BD:	Body	Body	text	M
:GB:	Gotten body	"	text	M
:W:	A form of <i>what</i> (1) string	N/A	:Z:/:M:/:t:/:l:	S
:A:	A form of <i>what</i> (1) string	N/A	:Z:/:Y:/:M:/:l:/:Z:	S
:Z:	<i>what</i> (1) string delimiter	N/A	@(#)	S
:F:	SCCS file name	N/A	text	S
:PN:	SCCS file path name	N/A	text	S

* :Dt: = :DT: :I: :D: :T: :P: :DS: :DP:

EXAMPLES

prs -d "Users and/or user IDs for :F: are:\n:UN:" s.file
may produce on the standard output:

Users and/or user IDs for s.file are:

xyz
131
abc

prs -d "Newest delta for pgm :M:: :I: Created :D: By :P:" -r s.file
may produce on the standard output:

Newest delta for pgm main.c: 3.7 Created 77/12/1 By cas

As a *special case*:

prs s.file

may produce on the standard output:

D 1.1 77/12/1 00:00:00 cas 1 000000/00000/00000
MRs:

bl78-12345
bl79-54321

COMMENTS:

this is the comment line for s.file initial delta

for each delta table entry of the "D" type. The only keyletter argument allowed to be used with the *special case* is the -a keyletter.

FILES

/tmp/pr????

SEE ALSO

admin(1), delta(1), get(1), help(1), sccsfile(5).

Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help*(1) for explanations.

NAME

ps - process status

SYNOPSIS

ps [alx] [namelist]

DESCRIPTION

Ps prints certain indicia about active processes. The **a** option asks for information about all processes with terminals (ordinarily only one's own processes are displayed); **x** asks even about processes with no terminal; **l** asks for a long listing. The short listing contains the process ID, tty letter, the cumulative execution time of the process and an approximation to the command line.

The long listing is columnar and contains

- F** Flags associated with the process. 01: in core; 02: system process; 04: locked in core (e.g. for physical I/O); 10: being swapped; 20: being traced by another process.
- S** The state of the process. 0: nonexistent; S: sleeping; W: waiting; R: running; I: intermediate; Z: terminated; T: stopped.
- UID** The user ID of the process owner.
- PID** The process ID of the process; as in certain cults it is possible to kill a process if you know its true name.
- PPID** The process ID of the parent process.
- CPU** Processor utilization for scheduling.
- PRI** The priority of the process; high numbers mean low priority.
- NICE** Used in priority computation.
- ADDR** The core address of the process if resident, otherwise the disk address.
- SZ** The size in units of 1k of the core image of the process.
- WCHAN** The event for which the process is waiting or sleeping; if blank, the process is running.
- TTY** The controlling tty for the process.
- TIME** The cumulative execution time for the process.

The command and its arguments.

A process that has exited and has a parent, but has not yet been waited for by the parent is marked <defunct>. Ps makes an educated guess as to the file name and arguments given when the process was created by examining core memory or the swap area. The method is inherently somewhat unreliable and in any event a process is entitled to destroy this information, so the names cannot be counted on too much.

If a second argument is given, it is taken to be the file containing the system's namelist, otherwise, /unix is assumed.

FILES

/unix system namelist

/dev/mem core memory
/dev/kmem alternate core file
/dev searched to find swap device and tty names

SEE ALSO

kill(1)

BUGS

Things can change while *ps* is running; the picture it gives is only a close approximation to reality.
Some data printed for defunct processes is irrelevant

NAME

pti - phototypesetter interpreter

SYNOPSIS

pti [file ...]

DESCRIPTION

Pti shows the commands in a stream from the standard output of *troff*(1) using *troff*'s *-t* option, interpreting them as they would act on the typesetter. Horizontal motions shows as counts in internal units and are marked with '<' and '>' indicating left and right motion. Vertical space is called *lead* and is also indicated.

SEE ALSO

troff(1)

BUGS

Too cryptic for normal users, who should use "*troff -a ...*".

NAME

ptx - permuted index

SYNOPSIS

ptx [option] ... [input [output]]

DESCRIPTION

Ptx generates a permuted index to file *input* on file *output* (standard input and output default). It has three phases: the first does the permutation, generating one line for each keyword in an input line. The keyword is rotated to the front. The permuted file is then sorted. Finally, the sorted lines are rotated so the keyword comes at the middle of the page. *Ptx* produces output in the form:

.xx "tail" "before keyword" "keyword and after" "head"

where .xx may be an *nroff* or *troff*(1) macro for user-defined formatting. The *before keyword* and *keyword and after* fields incorporate as much of the line as will fit around the keyword when it is printed at the middle of the page. *Tail* and *head*, at least one of which is an empty string "", are wrapped-around pieces small enough to fit in the unused space at the opposite end of the line. When original text must be discarded, '/' marks the spot.

The following options can be applied:

- f Fold upper and lower case letters for sorting.
- t Prepare the output for the phototypesetter; the default line length is 100 characters.
- w *n* Use the next argument, *n*, as the width of the output line. The default line length is 72 characters.
- g *n* Use the next argument, *n*, as the number of characters to allow for each gap among the four parts of the line as finally printed. The default gap is 3 characters.
- o only
Use as keywords only the words given in the *only* file.
- i ignore
Do not use as keywords any words given in the *ignore* file. If the -i and -o options are missing, use /usr/lib/eign as the *ignore* file.
- b break
Use the characters in the *break* file to separate words. In any case, tab, newline, and space characters are always used as break characters.
- r Take any leading nonblank characters of each input line to be a reference identifier (as to a page or chapter) separate from the text of the line. Attach that identifier as a 5th field on each output line.

The index for this manual was generated using *ptx*.

FILES

/bin/sort
/usr/lib/eign

EXAMPLE

The following is a reasonably-minimal set of *nroff* statements which will format *ptx* output reasonably; the style is similar to the index for this manual. The spacings may need adjustment for different output devices.

```
.de xx
.if !""\ $1" \ $1\ fI\ a\ fR\ $2\ t\ $3\ fI\ a\ fR\ t\ $5
.if !""\ $4" \ t\ $2\ t\ $3\ fI\ a\ fR\ $4\ t\ $5
.if !""\ $1\ $4" \ t\ $2\ t\ $3\ fI\ a\ fR\ t\ $5
..
.nf
.ta 3.7iR 3.95iL 7.65iR 7.8iR
```

BUGS

Line length counts do not account for overstriking or proportional spacing.

NAME

`pwd` - working directory name

SYNOPSIS

`pwd`

DESCRIPTION

Pwd prints the pathname of the working (current) directory.

SEE ALSO

`cd(1)`

THE FOLLOWING IS A SUMMARY OF THE RESULTS OF THE RESEARCH CONDUCTED BY THE RESEARCHER IN THE FIELD OF THE STUDY OF THE EFFECTS OF THE VIBRATION OF THE EARTH DURING THE EARTHQUAKE OF 1906.

THE RESEARCHER HAS FOUND THAT THE VIBRATION OF THE EARTH DURING THE EARTHQUAKE OF 1906 WAS OF THE TYPE OF THE VIBRATION OF THE EARTH DURING THE EARTHQUAKE OF 1906.

THE RESEARCHER HAS FOUND THAT THE VIBRATION OF THE EARTH DURING THE EARTHQUAKE OF 1906 WAS OF THE TYPE OF THE VIBRATION OF THE EARTH DURING THE EARTHQUAKE OF 1906.

THE RESEARCHER HAS FOUND THAT THE VIBRATION OF THE EARTH DURING THE EARTHQUAKE OF 1906 WAS OF THE TYPE OF THE VIBRATION OF THE EARTH DURING THE EARTHQUAKE OF 1906.

THE RESEARCHER HAS FOUND THAT THE VIBRATION OF THE EARTH DURING THE EARTHQUAKE OF 1906 WAS OF THE TYPE OF THE VIBRATION OF THE EARTH DURING THE EARTHQUAKE OF 1906.

NAME

ranlib - convert archives to random libraries

SYNOPSIS

ranlib archive ...

DESCRIPTION

Ranlib converts each *archive* to a form which can be loaded more rapidly by the loader, by adding a table of contents named `__SYMDEF` to the beginning of the archive. It uses *ar*(1) to reconstruct the archive, so that sufficient temporary file space must be available in the file system containing the current directory.

SEE ALSO

ld(1), ar(1)

BUGS

Because generation of a library by *ar* and randomization by *ranlib* are separate, phase errors are possible. The loader *ld* warns when the modification date of a library is more recent than the creation of its dictionary; but this means you get the warning even if you only copy the library.

NAME

ratfor - rational Fortran dialect

SYNOPSIS

ratfor [options] [files]

DESCRIPTION

Ratfor converts a rational dialect of Fortran into ordinary irrational Fortran. *Ratfor* provides control flow constructs essentially identical to those in C:

statement grouping:

```
{ statement; statement; statement }
```

decision-making:

```
if (condition) statement [ else statement ]
```

```
switch (integer value) {
```

```
    case integer: statement
```

```
    ...
```

```
    [ default: ] statement
```

```
}
```

loops:

```
while (condition) statement
```

```
for (expression; condition; expression) statement
```

```
do limits statement
```

```
repeat statement [ until (condition) ]
```

```
break
```

```
next
```

and some syntactic sugar to make programs easier to read and write:

free form input:

multiple statements/line; automatic continuation

comments:

```
# this is a comment.
```

translation of relationals:

>, >=, etc., become .GT., .GE., etc.

return expression to caller from function:

```
return (expression)
```

define:

```
define name replacement
```

include:

```
include file
```

The option **-h** causes quoted strings to be turned into **27H** constructs. The **-C** option copies comments to the output and attempts to format it neatly. Normally, continuation lines are marked with a **&** in column 1; the option **-6x** makes the continuation character **x** and places it in column 6.

Ratfor is best used with *f77(1)*.

SEE ALSO

efl(1), *f77(1)*.

B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

NAME

readnews - read news articles

SYNOPSIS

```
readnews [ -a date ] [ -n newsgroups ] [ -t titles ] [ -lprxhfum ] [
-c [ mailer ] ]
```

readnews -s

DESCRIPTION

readnews without argument prints unread articles. There are several interfaces available:

Flag	Interface
default	A <i>msgs(1)</i> like interface.
-M	An interface to <i>Mail(1)</i> .
-c	A <i>/bin/mail(1)</i> -like interface.
-c " <i>mailer</i> "	

All selected articles written to a temporary file. Then the mailer is invoked. The name of the temporary file is referenced with a "%". Thus, "mail -f %" will invoke mail on a temporary file consisting of all selected messages.

-p All selected articles are sent to the standard output. No questions asked.

-l Only the titles output. The *.newsrc* file will not be updated.

The -r flag causes the articles to be printed in reverse order. The -f flag prevents any followup articles from being printed. The -h flag causes articles to be printed in a less verbose format, and is intended for terminals running at 300 baud. the -u flag causes the *.newsrc* file to be updated every 5 minutes, in case of an unreliable system. (Note that if the *newsrc* file is updated, the x command will not restore it to its original contents.)

The following flags determine the selection of articles.

-n *newsgroups*

Select all articles that belong to *newsgroups*.

-t *titles* Select all articles whose titles contain one of the strings specified by *titles*.

-a [*date*]

Select all articles that were posted past the given *date* (in *getdate(3)* format).

-x

Ignore *.newsrc* file. That is, select articles that have already been read as well as new ones.

readnews maintains a *.newsrc* file in the user's home directory that specifies all news articles already read. It is updated at the end of each reading session in which the -x or -l options weren't specified. If the environment variable NEWSRC is present, it should be the path name of a file to be used in place of *.newsrc*.

If the user wishes, an options line may be placed in the *.newsrc* file. This line starts with the word **options** (left justified) followed by the list of

standard options just as they would be typed on the command line. Such a list may include: the `-n` flag along with a newsgroup list; a favorite interface; and/or the `-r` or `-t` flag. Continuation lines are specified by following lines beginning with a space or tab character. Similarly, options can be specified in the **NEWSOPTS** environment parameter. Where conflicts exist, option on the command line take precedence, followed by the `.newsrsrc` options line, and lastly the **NEWSOPTS** parameter.

readnews -s will print the newsgroup subscription list.

When the user uses the reply command of the `msgs(1)` or `/bin/mail(1)` interfaces, the environment parameter **MAILER** will be used to determine which mailer to use. The default is usually `/bin/mail`.

If the user so desires, he may specify a specific paging program for articles. The environment parameter **PAGER** should be set to the paging program. The name of the article is referenced with a `%`, as in the `-c` option. If no `%` is present, the article will be piped to the program. Paging may be disabled by setting **PAGER** to a null value.

COMMANDS

This section lists the commands you can type to the `msgs` and `/bin/mail` interface prompts. The `msgs` interface will suggest some common commands in brackets. Just hitting return is the same as typing the first command. For example, "[ynq]" means that the commands "y" (yes), "n" (no), and "q" (quit) are common responses, and that "y" is the default. Command Meaning

- | | |
|---------------|--|
| y | Yes. Prints current article and goes on to next. |
| n | No. Goes on to next article without printing current one. In the <code>/bin/mail</code> interface, this means "go on to the next article", which will have the same effect as "y" or just hitting return. |
| q | Quit. The <code>.newsrsrc</code> file will be updated if <code>-l</code> or <code>-x</code> were not on the command line. |
| c | Cancel the article. Only the author or the super user can do this. |
| r | Reply. Reply to article's author via mail. You are placed in your EDITOR with a header specifying To, Subject, and References lines taken from the message. You may change or add headers, as appropriate. You add the text of the reply after the blank line, and then exit the editor. The resulting message is mailed to the author of the article. |
| rd | Reply directly. You are placed in <code>\$MAILER</code> ("mail" by default) in reply to the author. Type the text of the reply and then control-Z. |
| f [title] | Submit a follow up article. Normally you should leave off the title, since the system will generate one for you. You will be placed in your EDITOR to compose the text of the followup. |
| fd | Followup directly, without edited headers. This is like <code>f</code> , but the headers of the article are not included in the editor buffer. |
| N [newsgroup] | Go to the next newsgroup or named newsgroup. |

s [*file*]

Save. The article is appended to the named file. The default is "Articles". If the first character of the file name is '|', the rest of the file name is taken as the name of a program, which is executed with the text of the article as standard input. If the first character of the file name is '/', it is taken as a full path name of a file. If \$NEWSBOX (in the environment) is set to a full path name, and the file contains no '/', the file is saved in \$NEWSBOX. Otherwise, it is saved relative to \$HOME.

- # Report the name and size of the newsgroup.
- e Erase. Forget that this article was read.
- h Print a more verbose header.
- H Print a very verbose header, containing all known information about the article.
- U Unsubscribe from this newsgroup. Also goes on to the next newsgroup.
- d Read a digest. Breaks up a digest into separate articles and permits you to read and reply to each piece.
- D Decrypt. Invokes a Caesar decoding program on the body of the message. This is used to decrypt rotated jokes posted to net.jokes. Such jokes are usually obscene or otherwise offensive to some groups of people, and so are rotated to avoid accidental decryption by people who would be offended. The title of the joke should indicate the nature of the problem, enabling people to decide whether to decrypt it or not.

Normally the Caesar program does a character frequency count on each line of the article separately, so that lines which are not rotated will be shown in plain text. This works well unless the line is short, in which case it sometimes gets the wrong rotation. An explicit *number* rotation (usually 13) may be given to force a particular shift.

- v Print the current version of the news software.
- ! Shell escape.

number

Go to *number*.

- +*[n]* Skip *n* articles. The articles skipped are recorded as "unread" and will be offered to you again the next time you read news.
- Go back to last article. This is a toggle, typing it twice returns you to the original article.
- x Exit. Like quit except that .newsrsrc is not updated.

X *system*

Transmit article to the named system.

The commands c, f, fd, r, rd, e, h, H, and s can be followed by -'s to refer to the previous article. Thus, when replying to an article using the msgs interface, you should normally type "r-" (or "re-") since by the time you enter a command, you are being offered the next article.

EXAMPLES

readnews

Read all unread articles using the *msgs(1)* interface. The *.newsrc* file is updated at the end of the session.

readnews -c "ed %" -l

Invoke the *ed(1)* text editor on a file containing the titles of all unread articles. The *.newsrc* file is **not** updated at the end of the session.

readnews -n all !fa.all -M -r

Read all unread articles except articles whose newsgroups begin with "fa." via *Mail(1)* in reverse order. The *.newsrc* file is updated at the end of the session.

readnews -p -n all -a last thursday

Print every unread article since last Thursday. The *.newsrc* file is updated at the end of the session.

readnews -p > /dev/null &

Discard all unread news. This is useful after returning from a long trip.

FILES

/usr/spool/news/newsgroup/number

News articles

/usr/lib/news/active Active newsgroups and numbers of articles

/usr/lib/news/help Help file for *msgs(1)* interface

~/.newsrc Options and list of previously read articles

SEE ALSO

checknews(1), *inews(1)*, *sendnews(8)*, *recnews(8)*, *uurec(8)*, *msgs(1)*, *Mail(1)*, *mail(1)*, *news(5)*, *newsrc(5)*

AUTHORS

Matt Glickman

Mark Horton

Stephen Daniel

Tom R. Truscott

NAME

recnews - receive unprocessed articles via mail

SYNOPSIS

recnews [newsgroup [sender]]

DESCRIPTION

recnews reads a letter from the standard input; determines the article title, sender, and newsgroup; and gives the body to *inews* with the right arguments for insertion.

If *newsgroup* is omitted, the to line of the letter will be used. If *sender* is omitted, the sender will be determined from the from line of the letter. The title is determined from the subject line.

SEE ALSO

inews(1), *uurec*(8), *sendnews*(8), *readnews*(1), *checknews*(1)

NAME

refer, lookbib - find and insert literature references in documents

SYNOPSIS

refer [option] ...

lookbib [file] ...

DESCRIPTION

Lookbib accepts keywords from the standard input and searches a bibliographic data base for references that contain those keywords anywhere in title, author, journal name, etc. Matching references are printed on the standard output. Blank lines are taken as delimiters between queries.

Refer is a preprocessor for *nroff* or *troff*(1) that finds and formats references. The input files (standard input default) are copied to the standard output, except for lines between *.[* and *.]* command lines, which are assumed to contain keywords as for *lookbib*, and are replaced by information from the bibliographic data base. The user may avoid the search, override fields from it, or add new fields. The reference data, from whatever source, are assigned to a set of *troff* strings. Macro packages such as *ms*(7) print the finished reference text from these strings. A flag is placed in the text at the point of reference; by default the references are indicated by numbers.

The following options are available:

- ar Reverse the first *r* author names (Jones, J. A. instead of J. A. Jones). If *r* is omitted all author names are reversed.
- b Bare mode: do not put any flags in text (neither numbers nor labels).
- cstring .
Capitalize (with CAPS SMALL CAPS) the fields whose key-letters are in *string*.
- e Instead of leaving the references where encountered, accumulate them until a sequence of the form
 .
 \$LIST\$
 .
is encountered, and then write out all references collected so far. Collapse references to the same source.
- kx Instead of numbering references, use labels as specified in a reference data line beginning %x; by default x is L.
- lm,n
Instead of numbering references, use labels made from the senior author's last name and the year of publication. Only the first *m* letters of the last name and the last *n* digits of the date are used. If either *m* or *n* is omitted the entire name or date respectively is used.
- p Take the next argument as a file of references to be searched. The default file is searched last.

-n Do not search the default file.

-skeys

Sort references by fields whose key-letters are in the *keys* string; permute reference numbers in text accordingly. Implies **-e**. The key-letters in *keys* may be followed by a number to indicate how many such fields are used, with **+** taken as a very large number. The default is **AD** which sorts on the senior author and then date; to sort, for example, on all authors and then title use **-sA+T**.

To use your own references, put them in the format described in *pubindex(1)*. They can be searched more rapidly by running *pubindex(1)* on them before using *refer*; failure to index results in a linear search.

When *refer* is used with *eqn*, *neqn* or *tbl*, *refer* should be first, to minimize the volume of data passed through pipes.

FILES

/usr/dict/papers directory of default publication lists and indexes
/usr/lib/refer directory of programs

SEE ALSO

NAME

reform - reformat text file

SYNOPSIS

reform [tabspec1 [tabspec2]] [+bn] [+en] [+f] [+in] [+mn] [+pn] [+s]
[+tn]

DESCRIPTION

Reform reads each line of the standard input file, reformats it, and then writes it to the standard output. Various combinations of reformatting operations can be selected, of which the most common involve rearrangement of tab characters. It is often used to trim trailing blanks, truncate lines to a specified length, or prepend blanks to lines.

Reform first scans its arguments, which may be given in any order. It then processes its input file, performing the following actions upon each line, in the order given:

- A line is read from the standard input.
- If *+s* is given, all characters up to the first tab are stripped off and saved for later addition to the end of the line. Presumably, these characters comprise an "SCCS SID" produced by *get(1)*.
- The line is expanded into a tabless form, by replacing tabs with blanks according to the *input* tab specification *tabspec1*.
- If *+pn* is given, *n* blanks are prepended to the line.
- If *+tn* is given, the line is truncated to a length of *n* characters.
- All trailing blanks are now removed.
- If *+en* is included, the line is extended out with blanks to the length of *n* characters.
- If *+s* is given, the previously-saved "SCCS SID" is added to the end of the line.
- If *+bn* is given, the *n* characters at the beginning of the line are converted to blanks, if and only if all of them are either digits or blanks.
- If *+mn* is included, the line is moved left, i.e., *n* characters are removed from the beginning of the line.
- The line is now contracted by replacing some blanks with tab characters according to the list of tabs indicated by the *output* tab specification *tabspec2*, and is written to the standard output file. Option *+i* controls the method of contraction (see below).

The various arguments accepted by *reform* are as follows:

tabspec1

describes the tab stops assumed for the input file. This tab specification may take on any of the forms described in *tabs(1)*. In addition, the operand *—* indicates that the tab specification is to be found in the first line read from the standard input. If no legal tab specification is found there, *-8* is assumed. If *tabspec1* is omitted entirely, *—* is assumed.

tabspec2

describes the tabs assumed for the output file. It is

interpreted in the same way as *tabspec1*, except that omission of *tabspec2* causes the value of *tabspec1* to be used for *tabspec2*.

The remaining arguments are all optional and may be used in any combination, although only a few combinations make much sense. Specifying an argument causes an action to be performed, as opposed to the usual default of not performing the action. Some options include numeric values, which also have default values. Option actions are applied to each line in the order described above. Any line length mentioned applies to the length of a line just before the execution of the option described, and the terminating new-line is never counted in the line length.

- +bn** causes the first *n* characters of a line to be converted to blanks, if and only if those characters include only blanks and digits. If *n* is omitted, the default value is 6, which is useful in deleting sequence numbers from COBOL programs.
- +en** causes each line shorter than *n* characters to be extended out with blanks to that length. Omitting *n* implies a default value of 72. This option is useful for those rare cases in which sequence numbers need to be added to an existing unnumbered file. The use of **\$** in editor regular expressions is more convenient if all lines have equal length, so that the user can issue editor commands such as:
s/\$00001000/
- +f** causes a format line to be written to the standard output, preceding any other lines written. See *fspec(5)* for details regarding format specifications. The format line is taken from *tabspec2*, i.e., the line normally appears as follows:
<:t-*tabspec2* d:> .

If *tabspec2* is of the form —*file-name* (i.e., an indirect reference to a tab specification in the first line of the named file), then that tab specification line is written to the standard output.

- +in** controls the technique used to compress interior blanks into tabs. Unless this option is specified, any sequence of 1 or more blanks may be converted to a single tab character if that sequence occurs just before a tab stop. This causes no problems for blanks that occur before the first nonblank character in a line, and it is always possible to convert the result back to an equivalent tabless form. However, occasionally an interior blank (one occurring after the first nonblank) is converted to a tab when this is not intended. For instance, this might occur in any program written in a language utilizing blanks as delimiters. Any single blank might be converted to a tab if it occurred just before a tab stop. Insertion or deletion of characters preceding such a tab may cause it to be interpreted in an unexpected way at a later time. If the **+i** option is used, no string of blanks may be converted to a tab unless there are *n* or more contiguous blanks. The default value is 2. Note that leading blanks are always converted to tabs when possible. It

is recommended that conversion of programs from non -UNIX to UNIX systems use this option.

- +mm** causes each line to be moved left *n* characters, with a default value of 6. This can be useful for crunching COBOL programs.
- +pn** causes *n* blanks to be prepended (default of 6 if *n* is omitted). This option is effectively the inverse of **+mm**, and is often useful for adjusting the position of *nroff*(1) output for terminals lacking both forms tractor positioning and a settable left margin.
- +s** is used with the **-m** option of *get*(1). The **-m** option causes *get* to prepend to each generated line the appropriate SCCS SID, followed by a tab. The **+s** option causes *reform* to remove the SID from the front of the line, save it, then add it later to the end of the line. Because **+e72** is implied by this option, the effect is to produce 80-character card images with SCCS SID in columns 73-80. Up to 8 characters of the SID are shown; if it is longer, the eighth character is replaced by * and any characters to the right of it are discarded.
- +tn** causes any line longer than *n* characters to be truncated to that length. If *n* is omitted, the length defaults to 72. Sequence numbers can thus be removed and any blanks immediately preceding them deleted.

The following illustrate typical uses of *reform*. The terms PWB and OBJECT below refer to UNIX and non- UNIX computer systems, respectively. Each arrow indicates the direction of conversion. The character ? indicates an arbitrary tab specification; see *tabs*(1) for descriptions of legal specifications.

OBJECT ---> PWB (i.e., manipulation of RJE output):

Note that files transferred by RJE from OBJECT to PWB materialize with format -8.

reform -8 -c +t +b +i <oldfile >newfile (into COBOL)

reform -8 -c3 +t +m +i <oldfile >newfile (into COBOL, crunched)

NOTE: -c3 is the preferred format COBOL; it uses the least disk space of the COBOL formats.

PWB ---> OBJECT (i.e., preparation of files for RJE submission):

reform ? -8 <oldfile >newfile (from arbitrary format into -8)

get -p -m sccsfile | *reform* +s | send ...

PWB ONLY (i.e., no involvement with other systems):

pr file | *reform* ? -0 <oldfile (print on terminal without hardware tabs)

reform ? -0 <oldfile >newfile (convert file to tabless format)

DIAGNOSTICS

All diagnostics are fatal, and the offending line is displayed following the message.

"line too long" a line exceeds 512 characters (in tabless form).

"not SCCS -m" a line does not have at least one tab when **+s** flag is used.

Any of the diagnostics of *tabs*(1) can also appear.

EXIT CODES

- 0 - normal
- 1 - any error

SEE ALSO

get(1), nroff(1), send(1C), tabs(1), fspec(5).

BUGS

Reform is aware of the meanings of backspaces and escape sequences, so that it can be used as a postprocessor for *nroff*. However, be warned that the *+e*, *+m*, and *+t* options only count characters, not positions. Anyone using these options on output containing backspaces or halflines will probably obtain unexpected results.

NAME

`regcmp` - regular expression compile

SYNOPSIS

`regcmp` [-] files

DESCRIPTION

Regcmp, in most cases, precludes the need for calling *regcmp* (see *regex(3X)*) from C programs. This saves on both execution time and program size. The command *regcmp* compiles the regular expressions in *file* and places the output in *file.i*. If the - option is used, the output will be placed in *file.c*. The format of entries in *file* is a name (C variable) followed by one or more blanks followed by a regular expression enclosed in double quotes. The output of *regcmp* is C source code. Compiled regular expressions are represented as **extern char** vectors. *File.i* files may thus be *included* into C programs, or *file.c* files may be compiled and later loaded. In the C program which uses the *regcmp* output, *regex(abc,line)* will apply the regular expression named *abc* to *line*. Diagnostics are self-explanatory.

EXAMPLES

```
name "[A-Za-z][A-Za-z0-9_]*"
telno "\({0,1}([2-9][01][1-9])$0\){0,1} *"
      "([2-9][0-9]{2})$1[-]{0,1}"
      "([0-9]{4})$2"
```

In the C program that uses the *regcmp* output,

```
regex(telno, line, area, exch, rest)
```

will apply the regular expression named *telno* to *line*.

SEE ALSO

regex(3X).

1950

1951

1952

The first of the year was a very busy one for the
company. The new product was introduced and
the sales were very good. The company was
able to increase its production and the
profits were also very good. The company
was able to expand its market and the
sales were also very good. The company
was able to increase its production and the
profits were also very good. The company
was able to expand its market and the
sales were also very good.

The second of the year was also a very busy one for the
company. The new product was introduced and
the sales were very good. The company was
able to increase its production and the
profits were also very good. The company
was able to expand its market and the
sales were also very good. The company
was able to increase its production and the
profits were also very good. The company
was able to expand its market and the
sales were also very good.

The third of the year was also a very busy one for the
company. The new product was introduced and
the sales were very good. The company was
able to increase its production and the
profits were also very good. The company
was able to expand its market and the
sales were also very good. The company
was able to increase its production and the
profits were also very good. The company
was able to expand its market and the
sales were also very good.

The fourth of the year was also a very busy one for the
company. The new product was introduced and
the sales were very good. The company was
able to increase its production and the
profits were also very good. The company
was able to expand its market and the
sales were also very good. The company
was able to increase its production and the
profits were also very good. The company
was able to expand its market and the
sales were also very good.

The fifth of the year was also a very busy one for the
company. The new product was introduced and
the sales were very good. The company was
able to increase its production and the
profits were also very good. The company
was able to expand its market and the
sales were also very good. The company
was able to increase its production and the
profits were also very good. The company
was able to expand its market and the
sales were also very good.

NAME

rev - reverse lines of a file

SYNOPSIS

rev [file] ...

DESCRIPTION

Rev copies the named files to the standard output, reversing the order of characters in every line. If no file is specified, the standard input is copied.

THE UNIVERSITY OF CHICAGO
LIBRARY
540 EAST 57TH STREET
CHICAGO, ILL. 60637
U.S.A.

NAME

rjestat - RJE status report and interactive status console

SYNOPSIS

rjestat [*host*]... [*-shost*] [*-chost cmd*]...

DESCRIPTION

Rjestat provides a method of determining the status of an RJE link and of simulating an IBM remote console (with UNIX features added). When invoked with no arguments, *rjestat* reports the current status of all the RJE links connected to the UNIX system. The options are:

host Print the status of the line to *host*. *Host* is the pseudonym for a particular IBM system. It can be any name that corresponds to one in the first column of the RJE configuration file.

-shost After all the arguments have been processed, start an interactive status console to *host*.

-chost cmd Interpret *cmd* as if it were entered in status console mode to *host*. See below for the proper format of *cmd*.

In status console mode, *rjestat* prompts with the host pseudonym followed by : whenever it is ready to accept a command. Commands are terminated with a new-line. A line that begins with ! is sent to the UNIX shell for execution. A line that begins with the letter q terminates *rjestat*. All other input lines are assumed to have the form:

ibmcmd [*redirect*]

Ibmcmd is any IBM JES or HASP command. Only the super-user or *rje* login can send commands other than display or inquiry commands. *Redirect* is a pipeline or a redirection to a file (e.g., "> file" or "| grep ..."). The IBM response is written to the pipeline or file. If *redirect* is not present, the response is written to the standard output of *rjestat*.

An interrupt signal (DEL or BREAK) will cancel the command in progress and cause *rjestat* to return to the command input mode.

EXAMPLE

The following command reports the status of all the card readers attached to host A, remote 5. JES2 is assumed.

```
rjestat -cA '$du,rmt5 | grep RD'
```

DIAGNOSTICS

The message "RJE error: ..." indicates that *rjestat* found an inconsistency in the RJE system. This may be transient but should be reported to the site administrator.

FILES

/usr/rje/lines RJE configuration file

resp host response file that exists in the RJE subsystem directory (e.g., /usr/rje1).

SEE ALSO

send(1C), rje(8).

OS/VS2 HASP II Version 4 Operator's Guide, IBM SRL #GC27-6993.

Operator's Library: OS/VS2 Reference (JES2), IBM SRL #GC38-0210.

1. The first part of the report deals with the general situation of the country and the progress of the work during the year.

2. The second part contains a detailed account of the work done in the various departments.

3. The third part gives a summary of the results of the work and a statement of the financial position.

4. The fourth part contains a list of the names of the persons who have been employed during the year.

5. The fifth part contains a list of the names of the persons who have been promoted during the year.

6. The sixth part contains a list of the names of the persons who have been dismissed during the year.

7. The seventh part contains a list of the names of the persons who have been transferred during the year.

8. The eighth part contains a list of the names of the persons who have been appointed during the year.

9. The ninth part contains a list of the names of the persons who have been retired during the year.

10. The tenth part contains a list of the names of the persons who have been deceased during the year.

...

NAME

rm, *rmdir* - remove files or directories

SYNOPSIS

rm [*-fri*] file ...

rmdir dir ...

DESCRIPTION

Rm removes the entries for one or more files from a directory. If an entry was the last link to the file, the file is destroyed. Removal of a file requires write permission in its directory, but neither read nor write permission on the file itself.

If a file has no write permission and the standard input is a terminal, its permissions are printed and a line is read from the standard input. If that line begins with *y* the file is deleted, otherwise the file remains. No questions are asked when the *-f* option is given or if the standard input is not a terminal.

If a designated file is a directory, an error comment is printed unless the optional argument *-r* has been used. In that case, *rm* recursively deletes the entire contents of the specified directory, and the directory itself.

If the *-i* (interactive) option is in effect, *rm* asks whether to delete each file, and, under *-r*, whether to examine each directory.

Rmdir removes entries for the named directories, which must be empty.

SEE ALSO

unlink(2).

DIAGNOSTICS

Generally self-explanatory. It is forbidden to remove the file *..* merely to avoid the antisocial consequences of inadvertently doing something like:

*rm -r .**

NAME

`rmidel` - remove a delta from an SCCS file

SYNOPSIS

`rmidel -rSID files`

DESCRIPTION

Rmidel removes the delta specified by the *SID* from each named SCCS file. The delta to be removed must be the newest (most recent) delta in its branch in the delta chain of each named SCCS file. In addition, the specified must *not* be that of a version being edited for the purpose of making a delta (i. e., if a *p-file* (see *get(1)*) exists for the named SCCS file, the specified must *not* appear in any entry of the *p-file*).

If a directory is named, *rmidel* behaves as though each file in the directory were specified as a named file, except that non-SCCS files (last component of the path name does not begin with *s.*) and unreadable files are silently ignored. If a name of `-` is given, the standard input is read; each line of the standard input is taken to be the name of an SCCS file to be processed; non-SCCS files and unreadable files are silently ignored.

The exact permissions necessary to remove a delta are documented in the *Source Code Control System User's Guide*. Simply stated, they are either (1) if you make a delta you can remove it; or (2) if you own the file and directory you can remove a delta.

FILES

x-file (see *delta(1)*)
z-file (see *delta(1)*)

SEE ALSO

delta(1), *get(1)*, *help(1)*, *prs(1)*, *sccsfile(5)*.

Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

Use *help(1)* for explanations.

1911-12

1911-12

1911-12

The first part of the year was spent in the field, collecting specimens and making observations on the habits of the various species of birds and mammals. The weather was generally favorable, but there were some periods of heavy rain which made the work difficult.

The second part of the year was spent in the laboratory, preparing the specimens and making the necessary measurements and calculations. The work was very tedious, but it was necessary to do it carefully in order to obtain accurate results.

The third part of the year was spent in writing up the results of the work and preparing the report. The work was very busy, but it was necessary to do it in order to complete the report on time.

The fourth part of the year was spent in the field, collecting specimens and making observations on the habits of the various species of birds and mammals. The weather was generally favorable, but there were some periods of heavy rain which made the work difficult.

The fifth part of the year was spent in the laboratory, preparing the specimens and making the necessary measurements and calculations. The work was very tedious, but it was necessary to do it carefully in order to obtain accurate results.

The sixth part of the year was spent in writing up the results of the work and preparing the report. The work was very busy, but it was necessary to do it in order to complete the report on time.

The seventh part of the year was spent in the field, collecting specimens and making observations on the habits of the various species of birds and mammals. The weather was generally favorable, but there were some periods of heavy rain which made the work difficult.

The eighth part of the year was spent in the laboratory, preparing the specimens and making the necessary measurements and calculations. The work was very tedious, but it was necessary to do it carefully in order to obtain accurate results.

The ninth part of the year was spent in writing up the results of the work and preparing the report. The work was very busy, but it was necessary to do it in order to complete the report on time.

NAME

rxctrl - floppy disk manipulating program

SYNOPSIS

rxctrl - [fsdiz] /dev/rrx?

DESCRIPTION

Rxctrl is used to give commands to the floppy drive:

- f Format data fields according to specified floppy density. This does not replace rxformat (see *format*(8)), which must be used to format new floppies.
- s Byte swapping on.
- d restore default setting: no byte swap, interleave of sectors on, first fill upper, then lower floppy side.

Unusual commands:

- i Interleave of sectors off.
- z Zigzag on: sequential blocks on different floppy sides.

FILES

/dev/rrx*

SEE ALSO

rx(4), format(8)

1000

1000

1000

1000

1000

1000

1000

1000

1000

1000

1000

1000

1000

NAME

sact - print current SCCS file editing activity

SYNOPSIS

sact files

DESCRIPTION

Sact informs the user of any impending deltas to a named SCCS file. This situation occurs when *get*(1) with the *-e* option has been previously executed without a subsequent execution of *delta*(1). If a directory is named on the command line, *sact* behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of *-* is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

The output for each named file consists of five fields separated by spaces.

- | | |
|---------|--|
| Field 1 | specifies the SID of a delta that currently exists in the SCCS file to which changes will be made to make the new delta. |
| Field 2 | specifies the SID for the new delta to be created. |
| Field 3 | contains the logname of the user who will make the delta (i.e. executed a <i>get</i> for editing). |
| Field 4 | contains the date that <i>get -e</i> was executed. |
| Field 5 | contains the time that <i>get -e</i> was executed. |

SEE ALSO

delta(1), *get*(1), *unget*(1).

DIAGNOSTICS

Use *help*(1) for explanations.

TO: DIRECTOR, NATIONAL SECURITY AGENCY

FROM: SAC, NEW YORK

SUBJECT: [Illegible]

[Illegible text block]

[Illegible text block]

[Illegible text block]

[Illegible text block]

NAME

sag - system activity graph

SYNOPSIS

sag [options]

DESCRIPTION

Sag graphically displays the system activity data stored in a binary data file by a previous *sar*(1) run. Any of the *sar* data items may be plotted singly, or in combination; as cross plots, or versus time. Simple arithmetic combinations of data may be specified. *Sag* invokes *sar* and finds the desired data by string-matching the data column header (run *sar* to see what's available). These *options* are passed thru to *sar*:

-s *time* Select data later than *time* in the form hh[:mm]. Default is 08:00.

-e *time* Select data up to *time*. Default is 18:00.

-i *sec* Select data at intervals as close as possible to *sec* seconds.

-f *file* Use *file* as the data source for *sar*. Default is the current daily data file /usr/adm/sa/sadd.

Other options:

-T *term*

Produce output suitable for terminal *term*. See *tplot*(1G) for known terminals. Default for *term* is **\$TERM**.

-x *spec*

x axis specification with *spec* in the form:
"name[*op* name]...[*lo* *hi*]"

-y *spec*

y axis specification with *spec* in the same form as above.

Name is either a string that will match a column header in the *sar* report, with an optional device name in square brackets, e.g., r+w/s[dsk-1], or an integer value. *Op* is + - * or / surrounded by blanks. Up to five names may be specified. Parentheses are not recognized. Contrary to custom, + and - have precedence over * and /. Evaluation is left to right. Thus A / A + B * 100 is evaluated (A/(A+B))*100, and A + B / C + D is (A+B)/(C+D). *Lo* and *hi* are optional numeric scale limits. If unspecified, they are deduced from the data.

A single *spec* is permitted for the x axis. If unspecified, *time* is used. Up to 5 *spec*'s separated by ; may be given for -y. Enclose the -x and -y arguments in "" if blanks or \<CR> are included. The -y default is:

-y "%usr 0 100; %usr + %sys 0 100; %usr + %sys + %wio 0 100"

EXAMPLES

To see today's CPU utilization:

```
sag
```

To see activity over 15 minutes of all disk drives:

```
TS=`date +%H:%M`
```

```
sar -o tempfile 60 15
```

```
TE=`date +%H:%M`
```

```
sag -f tempfile -s $TS -e $TE -y "r+w/s[dsk]"
```

FILES

/usr/adm/sa/sadd daily data file for day dd.

SEE ALSO

sar(1), tplot(1G).

NAME

sar - system activity reporter

SYNOPSIS

```
sar [-ubdycwaqvmA] [-o file] t [ n ]
sar [-ubdycwaqvmA] [-s time] [-e time] [-i sec] [-f file]
```

DESCRIPTION

Sar, in the first instance, samples cumulative activity counters in the operating system at *n* intervals of *t* seconds. If the *-o* option is specified, it saves the samples in *file* in binary format. The default value of *n* is 1. In the second instance, with no sampling interval specified, sar extracts data from a previously recorded *file*, either the one specified by *-f* option or, by default, the standard system activity daily data file */usr/adm/sa/sadd* for the current day *dd*. The starting and ending times of the report can be bounded via the *-s* and *-e time* arguments of the form *hh[:mm[:ss]]*. The *-i* option selects records at *sec* second intervals. Otherwise, all intervals found in the data file are reported.

In either case, subsets of data to be printed are specified by option:

- u Report CPU utilization (the default):
%usr, %sys, %wio, %idle - portion of time running in user mode, running in system mode, idle with some process waiting for block I/O, and otherwise idle.
- b Report buffer activity:
bread/s, bwrit/s - transfers per second of data between system buffers and disk or other block devices;
lread/s, lwrit/s - accesses of system buffers;
%rcache, %wcache - cache hit ratios, e. g., 1 - bread/lread;
pread/s, pwrit/s - transfers via raw (physical) device mechanism.
- d Report activity for each block device, e. g., disk or tape drive:
%busy, avque - portion of time device was busy servicing a transfer request, average number of requests outstanding during that time;
r+w/s, blks/s - number of data transfers from or to device, number of bytes transferred in 512 byte units;
await, avserv - average time in ms. that transfer requests wait idly on queue, and average time to be serviced (which for disks includes seek, rotational latency and data transfer times).
- y Report TTY device activity:
rawch/s, canch/s, outh/s - input character rate, input character rate processed by canon, output character rate;
rcvin/s, xmtin/s, mdmin/s - receive, transmit and modem interrupt rates.
- c Report system calls:
scall/s - system calls of all types;
sread/s, swrit/s, fork/s, exec/s - specific system calls;
rchar/s, wchar/s - characters transferred by read and write system calls.
- w Report system swapping and switching activity:
swpin/s, swpot/s, bswin/s, bswot/s - number of transfers and number of 512 byte units transferred for swapins (including initial loading of some programs) and swapouts;
pswch/s - process switches.

- a Report use of file access system routines:
iget/s, namei/s, dirblk/s.
- q Report average queue length while occupied, and % of time occupied:
runq-sz, %runocc - run queue of processes in memory and runnable;
swpq-sz, %swpocc - swap queue of processes swapped out but ready to run.
- v Report status of text, process, inode and file tables:
text-sz, proc-sz, inod-sz, file-sz - entries/size for each table, evaluated once at sampling point;
text-ov, proc-ov, inod-ov, file-ov - overflows occurring between sampling points.
- m Report message and semaphore activities:
msg/s, sema/s - primitives per second.
- A Report all data. Equivalent to -udqbwca yvm.

EXAMPLES

To see today's CPU activity so far:

```
sar
```

To watch CPU activity evolve for 10 minutes and save data:

```
sar -o temp 60 10
```

To later review disk and tape activity from that period:

```
sar -d -f temp
```

FILES

/usr/adm/sa/sadd daily data file, where *dd* are digits representing the day of the month.

SEE ALSO

sag(1G), sar(8).

NAME

`sccsdiff` - compare two versions of an SCCS file

SYNOPSIS

`sccsdiff -rSID1 -rSID2 [-p] [-sn] files`

DESCRIPTION

Sccsdiff compares two versions of an SCCS file and generates the differences between the two versions. Any number of SCCS files may be specified, but arguments apply to all files.

`-rSID?` *SID1* and *SID2* specify the deltas of an SCCS file that are to be compared. Versions are passed to *bdiff*(1) in the order given.

`-p` pipe output for each file through *pr*(1).

`-sn` *n* is the file segment size that *bdiff* will pass to *diff*(1). This is useful when *diff* fails due to a high system load.

FILES

`/tmp/get?????` Temporary files

SEE ALSO

bdiff(1), *get*(1), *help*(1), *pr*(1).

Source Code Control System User's Guide by L. E. Bonanni and C. A. Salemi.

DIAGNOSTICS

"*file*: No differences" If the two versions are the same.

Use *help*(1) for explanations.

Dear Sir,

I am writing to you regarding the matter of the

contract for the supply of goods to the

Government of the State of New South Wales.

I am sorry to hear that you are unable to

supply the goods at the time required.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

I am sure that you will be able to

supply the goods at a later date.

NAME

script - make typescript of terminal session

SYNOPSIS

script [-a] [file]

DESCRIPTION

Script makes a typescript of everything printed on your terminal. The typescript is written to *file*, or appended to *file* if the *-a* option is given. It can be sent to the line printer later with *lpr*. If no file name is given, the typescript is saved in the file *typescript*.

The script ends when the forked shell exits.

This program is useful when using a crt and a hard-copy record of the dialog is desired, as for a student handing in a program that was developed on a crt when hard-copy terminals are in short supply.

BUGS

Script places **everything** in the log file. This is not what the naive user expects.

1. The first part of the document is a letter from the President of the United States to the Congress, dated January 1, 1861. It is a very important document, as it sets out the President's policy for the new year.

2. The second part of the document is a report from the Secretary of the Treasury, dated January 1, 1861. It contains a detailed account of the financial state of the country at the beginning of the year.

3. The third part of the document is a report from the Secretary of the Interior, dated January 1, 1861. It contains a detailed account of the state of the interior of the country at the beginning of the year.

NAME

`sdiff` - side-by-side difference program

SYNOPSIS

`sdiff` [options ...] *file1 file2*

DESCRIPTION

Sdiff uses the output of *diff*(1) to produce a side-by-side listing of two files indicating those lines that are different. Each line of the two files is printed with a blank gutter between them if the lines are identical, a < in the gutter if the line only exists in *file1*, a > in the gutter if the line only exists in *file2*, and a | for lines that are different.

For example:

x		y
a		a
b	<	
c	<	
d		d
	>	c

The following options exist:

-w *n* Use the next argument, *n*, as the width of the output line. The default line length is 130 characters.

-l Only print the left side of any lines that are identical.

-s Do not print identical lines.

-o *output*

Use the next argument, *output*, as the name of a third file that is created as a user controlled merging of *file1* and *file2*. Identical lines of *file1* and *file2* are copied to *output*. Sets of differences, as produced by *diff*(1), are printed; where a set of differences share a common gutter character. After printing each set of differences, *sdiff* prompts the user with a % and waits for one of the following user-typed commands:

l	append the left column to the output file
r	append the right column to the output file
s	turn on silent mode; do not print identical lines
v	turn off silent mode
e l	call the editor with the left column
e r	call the editor with the right column
e b	call the editor with the concatenation of left and right
e	call the editor with a zero length file
q	exit from the program

On exit from the editor, the resulting file is concatenated on the end of the *output* file.

SEE ALSO

diff(1), ed(1).

NAME

sed - stream editor

SYNOPSIS

sed [-n] [-e script] [-f sfile] [files]

DESCRIPTION

Sed copies the named *files* (standard input default) to the standard output, edited according to a script of commands. The *-f* option causes the script to be taken from file *sfile*; these options accumulate. If there is just one *-e* option and no *-f* options, the flag *-e* may be omitted. The *-n* option suppresses the default output. A script consists of editing commands, one per line, of the following form:

[address [, address]] function [arguments]

In normal operation, *sed* cyclically copies a line of input into a *pattern space* (unless there is something left after a *D* command), applies in sequence all commands whose *addresses* select that pattern space, and at the end of the script copies the pattern space to the standard output (except under *-n*) and deletes the pattern space.

Some of the commands use a *hold space* to save all or part of the *pattern space* for subsequent retrieval.

An *address* is either a decimal number that counts input lines cumulatively across files, a *\$* that addresses the last line of input, or a context address, i.e., a */regular expression/* in the style of *ed(1)* modified thus:

In a context address, the construction *\?regular expression?*, where *?* is any character, is identical to */regular expression/*. Note that in the context address *\xabc\xdfx*, the second *x* stands for itself, so that the regular expression is *abcxdf*.

The escape sequence *\n* matches a new-line *embedded* in the pattern space.

A period *.* matches any character except the *terminal* new-line of the pattern space.

A command line with no addresses selects every pattern space.

A command line with one address selects each pattern space that matches the address.

A command line with two addresses selects the inclusive range from the first pattern space that matches the first address through the next pattern space that matches the second. (If the second address is a number less than or equal to the line number first selected, only one line is selected.) Thereafter the process is repeated, looking again for the first address.

Editing commands can be applied only to non-selected pattern spaces by use of the negation function *!* (below).

In the following list of functions the maximum number of permissible addresses for each function is indicated in parentheses.

The *text* argument consists of one or more lines, all but the last of which end with ** to hide the new-line. Backslashes in text are treated like backslashes in the replacement string of an *s* command, and may be used

to protect initial blanks and tabs against the stripping that is done on every script line. The *rfile* or *wfile* argument must terminate the command line and must be preceded by exactly one blank. Each *wfile* is created before processing begins. There can be at most 10 distinct *wfile* arguments.

- (1) *a* \
text Append. Place *text* on the output before reading the next input line.
- (2) *b label*
Branch to the : command bearing the *label*. If *label* is empty, branch to the end of the script.
- (2) *c* \
text Change. Delete the pattern space. With 0 or 1 address or at the end of a 2-address range, place *text* on the output. Start the next cycle.
- (2) *d* Delete the pattern space. Start the next cycle.
- (2) *D* Delete the initial segment of the pattern space through the first new-line. Start the next cycle.
- (2) *g* Replace the contents of the pattern space by the contents of the hold space.
- (2) *G* Append the contents of the hold space to the pattern space.
- (2) *h* Replace the contents of the hold space by the contents of the pattern space.
- (2) *H* Append the contents of the pattern space to the hold space.
- (1) *i* \
text Insert. Place *text* on the standard output.
- (2) *l* List the pattern space on the standard output in an unambiguous form. Non-printing characters are spelled in two-digit ASCII and long lines are folded.
- (2) *n* Copy the pattern space to the standard output. Replace the pattern space with the next line of input.
- (2) *N* Append the next line of input to the pattern space with an embedded new-line. (The current line number changes.)
- (2) *p* Print. Copy the pattern space to the standard output.
- (2) *P* Copy the initial segment of the pattern space through the first new-line to the standard output.
- (1) *q* Quit. Branch to the end of the script. Do not start a new cycle.
- (2) *r rfile*
Read the contents of *rfile*. Place them on the output before reading the next input line.
- (2) *s/regular expression/replacement/flags*
Substitute the *replacement* string for instances of the *regular expression* in the pattern space. Any character may be used instead of /. For a fuller description see *ed(1)*. *Flags* is zero or more of:
 - g* Global. Substitute for all nonoverlapping instances of the *regular expression* rather than just the first one.
 - p* Print the pattern space if a replacement was made.

w wfile

Write. Append the pattern space to *wfile* if a replacement was made.

(2) t label

Test. Branch to the : command bearing the *label* if any substitutions have been made since the most recent reading of an input line or execution of a **t**. If *label* is empty, branch to the end of the script.

(2) w wfile

Write. Append the pattern space to *wfile*.

(2) x Exchange the contents of the pattern and hold spaces.

(2) y/string1/string2/

Transform. Replace all occurrences of characters in *string1* with the corresponding character in *string2*. The lengths of *string1* and *string2* must be equal.

(2)! function

Don't. Apply the *function* (or group, if *function* is { }) only to lines *not* selected by the address(es).

(0) : label

This command does nothing; it bears a *label* for **b** and **t** commands to branch to.

(1) = Place the current line number on the standard output as a line.

(2) { Execute the following commands through a matching **}** only when the pattern space is selected.

(0) An empty command is ignored.

SEE ALSO

awk(1), ed(1), grep(1).

NAME

send - gather files and submit RJE jobs

SYNOPSIS

send argument ...

DESCRIPTION

Send is a command-level interface to the RJE subsystems. It allows the user to collect input from various sources in order to create a run stream consisting of card images, and submit this run stream for transmission to a host computer.

Possible sources of input to *send* are: ordinary files, standard input, the terminal, and the output of a command or shell file. Each source of input is treated as a virtual file, and no distinction is made based upon its origin. Typical input is an ASCII text file of the sort that is created by the editor *ed*(1). An optional format specification appearing in the first line of a file (see *fspec*(5)) determines the settings according to which tabs are expanded into spaces. In addition, lines that begin with ~ are normally interpreted as commands controlling the execution of *send*. They may be used to set or reset flags, to define keyword substitutions, and to open new sources of input in the midst of the current source. Other text lines are translated one-for-one into card images of the run stream.

The run stream that results from this collection is treated as one job by the RJE subsystems. *Send* prints the card count of the run stream, and the queuer that is invoked prints the name of the temporary file that holds the job while it is awaiting transmission. The initial card of a job submitted to an IBM host must have a // in the first column. The initial card of a job submitted to a UNIVAC host must begin with a "@RUN" or "~run", etc. Any cards preceding these will be excised. If a host computer is not specified before the first card of the runstream is ready to be sent, *send* will select a reasonable default. In the case of an IBM job, all cards beginning with /*\$ will be excised from the runstream, because they are HASP command cards.

The arguments that *send* accepts are described below. An argument is interpreted according to the first pattern that it matches. Preceding a character with \ causes it to lose any special meaning it might otherwise have when matching against an argument pattern.

.	Close the current source.
-	Open standard input as a new source.
+	Open the terminal as a new source.
:spec:	Establish a default format specification for included sources, e.g., :m6t-12:
:message	Print message on the terminal.
-:prompt	Open standard input and, if it is a terminal, print prompt.
+:prompt	Open the terminal and print prompt.

<i>-flags</i>	Set the specified flags, which are described below.
<i>+flags</i>	Reset the specified flags.
<i>=flags</i>	Restore the specified flags to their state at the previous level.
<i>!command</i>	Execute the specified UNIX <i>command</i> via the one-line shell, with input redirected to <i>/dev/null</i> as a default. Open the standard output of the command as a new source.
<i>\$line</i>	Collect contiguous arguments of this form and write them as consecutive lines to a temporary file; then have the file executed by the shell. Open the standard output of the shell as a new source.
<i>@directory</i>	The current directory for the send process is changed to <i>directory</i> . The original directory will be restored at the end of the current source.
<i>~comment</i>	Ignore this argument.
<i>?keyword</i>	Prompt for a definition of <i>keyword</i> from the terminal unless <i>keyword</i> has an existing definition.
<i>?keyword=xx</i>	Define the <i>keyword</i> as a two digit hexadecimal character code unless it already has a non null replacement.
<i>?keyword=string</i>	Define the <i>keyword</i> in terms of a replacement string unless it already has a non null replacement.
<i>=:keyword</i>	Prompt for a definition of <i>keyword</i> from the terminal.
<i>keyword=xx</i>	Define <i>keyword</i> as a two-digit hexadecimal character code.
<i>keyword=string</i>	Define <i>keyword</i> in terms of a replacement string.
<i>host</i>	The host machine that the job should be submitted to. It can be any name that corresponds to one in the first column of the RJE configuration file (<i>/usr/rje/lines</i>).
<i>file-name</i>	Open the specified file as a new source of input.

When commands are executed via *\$* or *!* the shell environment (see *environ(7)*) will contain the values of all send keywords that begin with *\$* and have the syntax of a shell variable.

The flags recognized by *send* are described in terms of the special processing that occurs when they are set:

- l* List card images on standard output. EBCDIC characters are translated back to ASCII.
- q* Do not output card images.
- f* Do not fold lower case to upper.
- t* Trace progress on diagnostic output, by announcing the opening of input sources.

- k Ignore the keywords that are active at the previous level and erase any keyword definitions that have been made at the current level.
- r Process included sources in raw mode; pack arbitrary 8-bit bytes one per column (80 columns per card) until an EOF.
- i Do not interpret control lines in included sources; treat them as text.
- s Make keyword substitutions before detecting and interpreting control lines.
- y Suppress error diagnostics and submit job anyway.
- g Gather mode, qualifying -l flag; list text lines before converting them to card images.
- h Write listing with standard tabs.
- p Prompt with * when taking input from the terminal.
- m When input returns to the terminal from a lower level, repeat the prompt, if any.
- a Make -k flag propagate to included sources, thereby protecting them from keyword substitutions.
- c List control lines on diagnostic output.
- d Extend the current set of keyword definitions by adding those active at the end of included sources.
- x This flag guarantees that the job will be transmitted in the order of submission (relative to other jobs sent with this flag).

Control lines are input lines that begin with ~. In the default mode +ir, they are interpreted as commands to *send*. Normally they are detected immediately and read literally. The -s flag forces keyword substitutions to be made before control lines are intercepted and interpreted. This can lead to unexpected results if a control line uses a keyword which is defined within an immediately preceding ~s sequence. Arguments appearing in control lines are handled exactly like the command arguments to *send*, except that they are processed at a nested level of input.

The two possible formats for a control line are: "~argument" and "~ argument ...". In the first case, where the ~ is not followed by a space, the remainder of the line is taken as a single argument to *send*. In the second case, the line is parsed to obtain a sequence of arguments delimited by spaces. In this case the quotes ' and " may be employed to pass embedded spaces.

The interpretation of the argument . is chosen so that an input line consisting of ~. is treated as a logical EOF. The following example illustrates some of the above conventions:

```
send -  
~ argument ...  
~.
```


This sequence of three lines is equivalent to the command synopsis at the beginning of this description. In fact, the `-` is not even required. By convention, the `send` command reads standard input if no other input source is specified. `Send` may therefore be employed as a filter with side-effects.

The execution of the `send` command is controlled at each instant by a current environment, which includes the format specification for the input source, a default format specification for included sources, the settings of the mode flags, and the active set of keyword definitions. This environment can be altered dynamically. When a control line opens a new source of input, the current environment is pushed onto a stack, to be restored when input resumes from the old source. The initial format specification for the new source is taken from the first line of the file. If none is provided, the established default is used or, in its absence, standard tabs. The initial mode settings and active keywords are copied from the old environment. Changes made while processing the new source will not affect the environment of the old source, with one exception: if `-d` mode is set in the old environment, the old keyword context will be augmented by those definitions that are active at the end of the new source.

When `send` first begins execution, all mode flags are reset, and the values of the shell environment variables become the initial values for keywords of the same name with a `$` prefixed.

The initial reset state for all mode flags is the `+` state. In general, special processing associated with a mode `N` is invoked by flag `-N` and is revoked by flag `+N`. Most mode settings have an immediate effect on the processing of the current source. Exceptions to this are the `-r` and `-i` flags, which apply only to included source, causing it to be processed in an uninterpreted manner.

A keyword is an arbitrary 8-bit ASCII string for which a replacement has been defined. The replacement may be another string, or (for IBM RJE only) the hexadecimal code for a single 8-bit byte. At any instant, a given set of keyword definitions is active. Input text lines are scanned, in one pass from left to right, and longest matches are attempted between substrings of the line and the active set of keywords. Characters that do not match are output, subject to folding and the standard translation. Keywords are replaced by the specified hexadecimal code or replacement string, which is then output character by character. The expansion of tabs and length checking, according to the format specification of an input source, are delayed until substitutions have been made in a line.

All of the keywords definitions made in the current source may be deleted by setting the `-k` flag. It then becomes possible to reuse them. Setting the `-k` flag also causes keyword definitions active at the previous source level to be ignored. Setting the `+k` flag causes keywords at the previous level to be ignored but does not delete the definitions made at the current level. The `=k` argument reactivates the definitions of the previous level.

When keywords are redefined, the previous definition at the same level of source input is lost, however the definition at the previous level is only hidden, to be reactivated upon return to that level unless a `-d` flag causes the current definition to be retained.

Conditional prompts for keywords, `?A,/p` which have already been defined at some higher level to be null or have a replacement will simply cause the definitions to be copied down to the current level; new definitions will not be solicited.

Keyword substitution is an elementary macro facility that is easily explained and that appears useful enough to warrant its inclusion in the `send` command. More complex replacements are the function of a general macro processor (*m4*(1), perhaps). To reduce the overhead of string comparison, it is recommended that keywords be chosen so that their initial characters are unusual. For example, let them all be upper case.

`Send` performs two types of error checking on input text lines. Firstly, only ASCII, graphics and tabs are permitted in input text. Secondly, the length of a text line, after substitutions have been made, may not exceed 80 bytes for IBM, or 132 bytes for UNIVAC. The length of each line may be additionally constrained by a size parameter in the format specification for an input source. Diagnostic output provides the location of each erroneous line, by line number and input source, a description of the error, and the card image that results. Other routine errors that are announced are the inability to open or write files, and abnormal exits from the shell. Normally, the occurrence of any error causes `send`, before invoking the queuer, to prompt for positive affirmation that the suspect run stream should be submitted.

For IBM hosts, `send` is required to translate 8-bit ASCII characters into their EBCDIC equivalents. The conversion for 8-bit ASCII characters in the octal range 040-176 is based on the character set described in "Appendix H" of *IBM System/370 Principles of Operation* (IBM SRL GA22-7000). Each 8-bit ASCII character in the range 040-377 possesses an EBCDIC equivalent into which it is mapped, with five exceptions: `~` into `~`, 0345 into `~`, 0325 into `¢`, 0313 into `|`, 0177 (DEL) is illegal. In listings requested from `send` and in printed output returned by the subsystem, the reverse translation is made with the qualification that EBCDIC characters that do not have valid 8-bit ASCII equivalents are translated into `~`. UNIVAC hosts, on the other hand, operate in ASCII code, and any translations between ASCII and field-data are made, in accordance with the UNIVAC standard, by the host computer.

Additional control over the translation process is afforded by the `-f` flag and hexadecimal character codes. As a default, `send` folds lower-case letters into upper case. For UNIVAC RJE it does more: the entire ASCII range 0140-0176 is folded into 0100-0136, so that `^`, for example, becomes `@`. In either case, setting the `-f` flag inhibits any folding. Non-standard character codes are obtained as a special case of keyword substitution.

SEE ALSO

m4(1), rjstat(1C), sh(1), fspec(5), ascii(7), rje(8).

Guide to IBM Remote Job Entry for PWB/UNIX Users by A. L. Sabsevit and E. J. Finger.

UNIX Remote Job Entry User's Guide by K. A. Kelleman.

BUGS

Standard input is read in blocks, and unused bytes are returned via `lseek(2)`. If standard input is a pipe, multiple arguments of the form `-` and `-prompt` should not be used, nor should the logical EOF (`~`).

NAME

sh, rsh - shell, the standard/restricted command programming language

SYNOPSIS

```
sh [ -ceiknrstuvx ] [ args ]
rsh [ -ceiknrstuvx ] [ args ]
```

DESCRIPTION

Sh is a command programming language that executes commands read from a terminal or a file. *Rsh* is a restricted version of the standard command interpreter *sh*; it is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. See *Invocation* below for the meaning of arguments to the shell.

Commands.

A *simple-command* is a sequence of non-blank *words* separated by *blanks* (a *blank* is a tab or a space). The first word specifies the name of the command to be executed. Except as specified below, the remaining words are passed as arguments to the invoked command. The command name is passed as argument 0 (see *exec(2)*). The *value* of a simple-command is its exit status if it terminates normally, or (octal) 200+*status* if it terminates abnormally (see *signal(2)* for a list of status values).

A *pipeline* is a sequence of one or more *commands* separated by | (or, for historical compatibility, by ~). The standard output of each command but the last is connected by a *pipe(2)* to the standard input of the next command. Each command is run as a separate process; the shell waits for the last command to terminate.

A *list* is a sequence of one or more pipelines separated by ;, &, &&, or ||, and optionally terminated by ; or &. Of these four symbols, ; and & have equal precedence, which is lower than that of && and ||. The symbols && and || also have equal precedence. A semicolon (;) causes sequential execution of the preceding pipeline; an ampersand (&) causes asynchronous execution of the preceding pipeline (i.e., the shell does *not* wait for that pipeline to finish). The symbol && (||) causes the *list* following it to be executed only if the preceding pipeline returns a zero (non-zero) exit status. An arbitrary number of new-lines may appear in a *list*, instead of semicolons, to delimit commands.

A *command* is either a simple-command or one of the following. Unless otherwise stated, the value returned by a command is that of the last simple-command executed in the command.

for name [in word ...] do list done

Each time a **for** command is executed, *name* is set to the next *word* taken from the *in word* list. If *in word ...* is omitted, then the **for** command executes the *do list* once for each positional parameter that is set (see *Parameter Substitution* below). Execution ends when there are no more words in the list.

case word in [pattern [|pattern] ...) list ;;] ... esac

A **case** command executes the *list* associated with the first *pattern* that matches *word*. The form of the patterns is the same as that used for file-name generation (see *File Name Generation* below).

if *list* **then** *list* [**elif** *list* **then** *list*] ... [**else** *list*] **fi**

The *list* following **if** is executed and, if it returns a zero exit status, the *list* following the first **then** is executed. Otherwise, the *list* following **elif** is executed and, if its value is zero, the *list* following the next **then** is executed. Failing that, the **else** *list* is executed. If no **else** *list* or **then** *list* is executed, then the **if** command returns a zero exit status.

while *list* **do** *list* **done**

A **while** command repeatedly executes the *while* *list* and, if the exit status of the last command in the *list* is zero, executes the *do* *list*; otherwise the loop terminates. If no commands in the *do* *list* are executed, then the **while** command returns a zero exit status; **until** may be used in place of **while** to negate the loop termination test.

(*list*)

Execute *list* in a sub-shell.

{*list*;}

list is simply executed.

The following words are only recognized as the first word of a command and when not quoted:

if then else elif fi case esac for while until do done { }

Comments.

A word beginning with **#** causes that word and all the following characters up to a new-line to be ignored.

Command Substitution.

The standard output from a command enclosed in a pair of grave accents (**`**) may be used as part or all of a word; trailing new-lines are removed.

Parameter Substitution.

The character **\$** is used to introduce substitutable *parameters*. Positional parameters may be assigned values by **set**. Variables may be set by writing:

name=*value* [*name*=*value*] ...

Pattern-matching is not performed on *value*.

\${parameter}

A *parameter* is a sequence of letters, digits, or underscores (a *name*), a digit, or any of the characters *****, **@**, **#**, **?**, **-**, **\$**, and **!**. The value, if any, of the parameter is substituted. The braces are required only when *parameter* is followed by a letter, digit, or underscore that is not to be interpreted as part of its name. A *name* must begin with a letter or underscore. If *parameter* is a digit then it is a positional parameter. If *parameter* is ***** or **@**, then all the positional parameters, starting with **\$1**, are substituted (separated by spaces). Parameter **\$0** is set from argument zero when the shell is invoked.

\${parameter:-word}

If *parameter* is set and is non-null then substitute its value; otherwise substitute *word*.

\${parameter:=word}

If *parameter* is not set or is null then set it to *word*; the value of the parameter is then substituted. Positional parameters may not

be assigned to in this way.

\${parameter:?word}

If *parameter* is set and is non-null then substitute its value; otherwise, print *word* and exit from the shell. If *word* is omitted, then the message "parameter null or not set" is printed.

\${parameter:+word}

If *parameter* is set and is non-null then substitute *word*; otherwise substitute nothing.

In the above, *word* is not evaluated unless it is to be used as the substituted string, so that, in the following example, **pwd** is executed only if *d* is not set or is null:

```
echo ${d:-`pwd`}
```

If the colon (:) is omitted from the above expressions, then the shell only checks whether *parameter* is set or not.

The following parameters are automatically set by the shell:

- #** The number of positional parameters in decimal.
- Flags supplied to the shell on invocation or by the **set** command.
- ?** The decimal value returned by the last synchronously executed command.
- \$** The process number of this shell.
- !** The process number of the last background command invoked.

The following parameters are used by the shell:

- HOME** The default argument (home directory) for the **cd** command.
- PATH** The search path for commands (see *Execution* below). The user may not change **PATH** if executing under *rsh*.
- CDPATH** The search path for the **cd** command.
- MAIL** If this variable is set to the name of a mail file, then the shell informs the user of the arrival of mail in the specified file.
- PS1** Primary prompt string, by default "**\$** ".
- PS2** Secondary prompt string, by default "> ".
- IFS** Internal field separators, normally **space**, **tab**, and **new-line**.

The shell gives default values to **PATH**, **PS1**, **PS2**, and **IFS**, while **HOME** and **MAIL** are not set at all by the shell (although **HOME** is set by *login*(1)).

Blank Interpretation.

After parameter and command substitution, the results of substitution are scanned for internal field separator characters (those found in **IFS**) and split into distinct arguments where such characters are found. Explicit null arguments (" " or "") are retained. Implicit null arguments (those resulting from *parameters* that have no values) are removed.

File Name Generation.

Following substitution, each command *word* is scanned for the characters *****, **?**, and **[**. If one of these characters appears then the word is regarded as a *pattern*. The word is replaced with alphabetically sorted file names that match the pattern. If no file name is found that matches

the pattern, then the word is left unchanged. The character `.` at the start of a file name or immediately following a `/`, as well as the character `/` itself, must be matched explicitly.

- * Matches any string, including the null string.
- ? Matches any single character.
- [...] Matches any one of the enclosed characters. A pair of characters separated by `-` matches any character lexically between the pair, inclusive. If the first character following the opening `[` is a `!` then any character not enclosed is matched.

Quoting.

The following characters have a special meaning to the shell and cause termination of a word unless quoted:

`; & () | ^ < > new-line space tab`

A character may be *quoted* (i.e., made to stand for itself) by preceding it with a `\`. The pair `\new-line` is ignored. All characters enclosed between a pair of single quote marks (`' '`), except a single quote, are quoted. Inside double quote marks (`" "`), parameter and command substitution occurs and `\` quotes the characters `\`, `'`, `"`, and `$`. `"$"` is equivalent to `"$1 $2 ..."`, whereas `"$@"` is equivalent to `"$1" "$2"`

Prompting.

When used interactively, the shell prompts with the value of `PS1` before reading a command. If at any time a new-line is typed and further input is needed to complete a command, then the secondary prompt (i.e., the value of `PS2`) is issued.

Input/Output.

Before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. The following may appear anywhere in a simple-command or may precede or follow a *command* and are *not* passed on to the invoked command; substitution occurs before *word* or *digit* is used:

- | | |
|------------------------------|--|
| <code><word</code> | Use file <i>word</i> as standard input (file descriptor 0). |
| <code>>word</code> | Use file <i>word</i> as standard output (file descriptor 1). If the file does not exist then it is created; otherwise, it is truncated to zero length. |
| <code>>>word</code> | Use file <i>word</i> as standard output. If the file exists then output is appended to it (by first seeking to the end-of-file); otherwise, the file is created. |
| <code><<[-]word</code> | The shell input is read up to a line that is the same as <i>word</i> , or to an end-of-file. The resulting document becomes the standard input. If any character of <i>word</i> is quoted, then no interpretation is placed upon the characters of the document; otherwise, parameter and command substitution occurs, (unescaped) <code>\new-line</code> is ignored, and <code>\</code> must be used to quote the characters <code>\</code> , <code>\$</code> , <code>'</code> , and the first character of <i>word</i> . If <code>-</code> is appended to <code><<</code> , then all leading tabs are stripped from <i>word</i> and from the document. |
| <code><&digit</code> | The standard input is duplicated from file descriptor <i>digit</i> (see <i>dup(2)</i>). Similarly for the standard output using <code>></code> . |

<&- The standard input is closed. Similarly for the standard output using >.

If one of the above is preceded by a digit, then the file descriptor created is that specified by the digit (instead of the default 0 or 1). For example:

```
... 2>&1
```

creates file descriptor 2 that is a duplicate of file descriptor 1.

If a command is followed by & then the default standard input for the command is the empty file `/dev/null`. Otherwise, the environment for the execution of a command contains the file descriptors of the invoking shell as modified by input/output specifications.

Redirection of output is not allowed in the restricted shell.

Environment.

The *environment* (see *environ(7)*) is a list of name-value pairs that is passed to an executed program in the same way as a normal argument list. The shell interacts with the environment in several ways. On invocation, the shell scans the environment and creates a parameter for each name found, giving it the corresponding value. Executed commands inherit the same environment. If the user modifies the values of these parameters or creates new ones, none of these affects the environment unless the **export** command is used to bind the shell's parameter to the environment. The environment seen by any executed command is thus composed of any unmodified name-value pairs originally inherited by the shell, plus any modifications or additions, all of which must be noted in **export** commands.

The environment for any *simple-command* may be augmented by prefixing it with one or more assignments to parameters. Thus:

```
TERM=450 cmd args          and
(export TERM; TERM=450; cmd args)
```

are equivalent (as far as the above execution of *cmd* is concerned).

If the **-k** flag is set, *all* keyword arguments are placed in the environment, even if they occur after the command name. The following first prints **a=b c** and then **c**:

```
echo a=b c
set -k
echo a=b c
```

Signals.

The **INTERRUPT** and **QUIT** signals for an invoked command are ignored if the command is followed by &; otherwise signals have the values inherited by the shell from its parent, with the exception of signal 11 (but see also the **trap** command below).

Execution.

Each time a command is executed, the above substitutions are carried out. Except for the *Special Commands* listed below, a new process is created and an attempt is made to execute the command via *exec(2)*.

The shell parameter **PATH** defines the search path for the directory containing the command. Alternative directory names are separated by a

colon (:). The default path is `:/bin:/usr/bin` (specifying the current directory, `/bin`, and `/usr/bin`, in that order). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If the command name contains a `/` then the search path is not used; such commands will not be executed by the restricted shell. Otherwise, each directory in the path is searched for an executable file. If the file has execute permission but is not an `a.out` file, it is assumed to be a file containing shell commands. A sub-shell (i.e., a separate process) is spawned to read it. A parenthesized command is also executed in a sub-shell.

Special Commands.

The following commands are executed in the shell process and, except as specified, no input/output redirection is permitted for such commands:

- `:` No effect; the command does nothing. A zero exit code is returned.
- `. file` Read and execute commands from *file* and return. The search path specified by `PATH` is used to find the directory containing *file*.
- `break [n]`
Exit from the enclosing **for** or **while** loop, if any. If *n* is specified then break *n* levels.
- `continue [n]`
Resume the next iteration of the enclosing **for** or **while** loop. If *n* is specified then resume at the *n*-th enclosing loop.
- `cd [arg]`
Change the current directory to *arg*. The shell parameter `HOME` is the default *arg*. The shell parameter `CDPATH` defines the search path for the directory containing *arg*. Alternative directory names are separated by a colon (:). The default path is `<null>` (specifying the current directory). Note that the current directory is specified by a null path name, which can appear immediately after the equal sign or between the colon delimiters anywhere else in the path list. If *arg* begins with a `/` then the search path is not used. Otherwise, each directory in the path is searched for *arg*. The `cd` command may not be executed by *rsh*.
- `eval [arg ...]`
The arguments are read as input to the shell and the resulting command(s) executed.
- `exec [arg ...]`
The command specified by the arguments is executed in place of this shell without creating a new process. Input/output arguments may appear and, if no other arguments are given, cause the shell input/output to be modified.
- `exit [n]`
Causes a shell to exit with the exit status specified by *n*. If *n* is omitted then the exit status is that of the last command executed (an end-of-file will also cause the shell to exit.)
- `export [name ...]`
The given *names* are marked for automatic export to the *environment* of subsequently-executed commands. If no arguments are given, then a list of all names that are exported in this shell is

printed.

newgrp [*arg* ...]

Equivalent to **exec newgrp arg**

read [*name* ...]

One line is read from the standard input and the first word is assigned to the first *name*, the second word to the second *name*, etc., with leftover words assigned to the last *name*. The return code is 0 unless an end-of-file is encountered.

readonly [*name* ...]

The given *names* are marked *readonly* and the values of these *names* may not be changed by subsequent assignment. If no arguments are given, then a list of all *readonly* names is printed.

set [**—ekntuvx** [*arg* ...]]

—e Exit immediately if a command exits with a non-zero exit status.

—k All keyword arguments are placed in the environment for a command, not just those that precede the command name.

—n Read commands but do not execute them.

—t Exit after reading and executing one command.

—u Treat unset variables as an error when substituting.

—v Print shell input lines as they are read.

—x Print commands and their arguments as they are executed.

— Do not change any of the flags; useful in setting **\$1** to **—**.

Using **+** rather than **—** causes these flags to be turned off. These flags can also be used upon invocation of the shell. The current set of flags may be found in **\$—**. The remaining arguments are positional parameters and are assigned, in order, to **\$1**, **\$2**, If no arguments are given then the values of all names are printed.

shift [*n*]

The positional parameters from **\$n+1** ... are renamed **\$1** If *n* is not given, it is assumed to be 1.

test

Evaluate conditional expressions. See **test(1)** for usage and description.

times

Print the accumulated user and system times for processes run from the shell.

trap [*arg*] [*n*] ...

arg is a command to be read and executed when the shell receives signal(s) *n*. (Note that *arg* is scanned once when the trap is set and once when the trap is taken.) Trap commands are executed in order of signal number. Any attempt to set a trap on a signal that was ignored on entry to the current shell is ineffective. An attempt to trap on signal 11 (memory fault) produces an error. If *arg* is absent then all trap(s) *n* are reset to their original values. If *arg* is the null string then this signal is ignored by the shell and by the commands it invokes. If *n* is 0 then the command *arg* is executed on exit from the shell. The **trap** command with no arguments prints a list of commands associated with each signal number.

ulimit [**—fp**] [*n*]

imposes a size limit of *n*

-f imposes a size limit of *n* blocks on files written by child processes (files of any size may be read). With no argument, the current limit is printed.

-p changes the pipe size to *n* (UNIX System/RT only).

If no option is given, **-f** is assumed.

umask [*nnn*]

The user file-creation mask is set to *nnn* (see *umask(2)*). If *nnn* is omitted, the current value of the mask is printed.

wait [*n*]

Wait for the specified process and report its termination status. If *n* is not given then all currently active child processes are waited for and the return code is zero.

Invocation.

If the shell is invoked through *exec(2)* and the first character of argument zero is **-**, commands are initially read from */etc/profile* and then from *\$HOME/.profile*, if such files exist. Thereafter, commands are read as described below, which is also the case when the shell is invoked as */bin/sh*. The flags below are interpreted by the shell on invocation only; Note that unless the **-c** or **-s** flag is specified, the first argument is assumed to be the name of a file containing commands, and the remaining arguments are passed as positional parameters to that command file:

-c *string*

If the **-c** flag is present then commands are read from *string*.

-s

If the **-s** flag is present or if no arguments remain then commands are read from the standard input. Any remaining arguments specify the positional parameters. Shell output is written to file descriptor 2.

-i

If the **-i** flag is present or if the shell input and output are attached to a terminal, then this shell is *interactive*. In this case **TERMINATE** is ignored (so that **kill 0** does not kill an interactive shell) and **INTERRUPT** is caught and ignored (so that **wait** is interruptible). In all cases, **QUIT** is ignored by the shell.

-r

If the **-r** flag is present the shell is a restricted shell.

The remaining flags and arguments are described under the **set** command above.

Rsh Only.

Rsh is used to set up login names and execution environments whose capabilities are more controlled than those of the standard shell. The actions of *rsh* are identical to those of *sh*, except that the following are disallowed:

changing directory (see *cd(1)*),

setting the value of *\$PATH*,

specifying path or command names containing */*,

redirecting output (*>* and *>>*).

The restrictions above are enforced after *.profile* is interpreted.

When a command to be executed is found to be a shell procedure, *rsh* invokes *sh* to execute it. Thus, it is possible to provide to the end-user shell procedures that have access to the full power of the standard shell, while imposing a limited menu of commands; this scheme assumes that

the end-user does not have write and execute permissions in the same directory.

The net effect of these rules is that the writer of the **.profile** has complete control over user actions, by performing *guaranteed setup* actions and leaving the user in an appropriate directory (probably *not* the login directory).

The system administrator often sets up a directory of commands (i.e., **/usr/rbin**) that can be safely invoked by *rsh*. Some systems also provide a restricted editor *red*.

EXIT STATUS

Errors detected by the shell, such as syntax errors, cause the shell to return a non-zero exit status. If the shell is being used non-interactively then execution of the shell file is abandoned. Otherwise, the shell returns the exit status of the last command executed (see also the **exit** command above).

FILES

/etc/profile
\$HOME/.profile
/tmp/sh*
/dev/null

SEE ALSO

cd(1), **env(1)**, **login(1)**, **newgrp(1)**, **test(1)**, **umask(1)**, **dup(2)**, **exec(2)**, **fork(2)**, **pipe(2)**, **signal(2)**, **ulimit(2)**, **umask(2)**, **wait(2)**, **a.out(5)**, **profile(5)**, **environ(7)**.

BUGS

The command **readonly** (without arguments) produces the same output as the command **export**.

If **<<** is used to provide standard input to an asynchronous process invoked by **&**, the shell gets mixed up about naming the input document; a garbage file **/tmp/sh*** is created and the shell complains about not being able to find that file by another name.

The first of these is the fact that the
government has been unable to
obtain the necessary funds to
carry out its policy. This is
due to the fact that the
government has been unable to
obtain the necessary funds to
carry out its policy.

The second of these is the fact that
the government has been unable to
obtain the necessary funds to
carry out its policy.

The third of these is the fact that
the government has been unable to
obtain the necessary funds to
carry out its policy.

The fourth of these is the fact that
the government has been unable to
obtain the necessary funds to
carry out its policy.

The fifth of these is the fact that
the government has been unable to
obtain the necessary funds to
carry out its policy.

NAME

size — size of an object file

SYNOPSIS

size [object ...]

DESCRIPTION

Size prints the (decimal) number of bytes required by the text, data, bss and stack portions, and their sum in octal and decimal, of each object-file argument. If no file is specified, **a.out** is used.

SEE ALSO

a.out(5)

1972

1973

1974

1975

1976

1977

1978

1979

1980

NAME

sleep — suspend execution for an interval

SYNOPSIS

sleep time

DESCRIPTION

Sleep suspends execution for *time* seconds. It is used to execute a command after a certain amount of time as in:

```
(sleep 105; command)&
```

or to execute a command every so often, as in:

```
while true
do
    command
    sleep 37
done
```

SEE ALSO

alarm(2), sleep(3)

BUGS

Time must be less than 65536 seconds.

MEMORANDUM FOR THE RECORD

SUBJECT: [Illegible]

[Illegible text]

[Illegible text]

[Illegible text]

[Illegible text]

[Illegible text]

NAME

sno - SNOBOL interpreter

SYNOPSIS

sno [files]

DESCRIPTION

Sno is a SNOBOL compiler and interpreter (with slight differences). *Sno* obtains input from the concatenation of the named *files* and the standard input. All input through a statement containing the label **end** is considered program and is compiled. The rest is available to **syspit**.

Sno differs from SNOBOL in the following ways:

There are no unanchored searches. To get the same effect:

a ** b unanchored search for b.

a *x* b = x cunanchored assignment

There is no back referencing.

x = "abc"

a *x* x is an unanchored search for abc.

Function declaration is done at compile time by the use of the (non-unique) label **define**. Execution of a function call begins at the statement following the **define**. Functions cannot be defined at run time, and the use of the name **define** is preempted. There is no provision for automatic variables other than parameters. Examples:

```
define f( )  
define f(a, b, c)
```

All labels except **define** (even **end**) must have a non-empty statement.

Labels, functions and variables must all have distinct names. In particular, the non-empty statement on **end** cannot merely name a label.

If **start** is a label in the program, program execution will start there. If not, execution begins with the first executable statement; **define** is not an executable statement.

There are no builtin functions.

Parentheses for arithmetic are not needed. Normal precedence applies. Because of this, the arithmetic operators / and * must be set off by spaces.

The right side of assignments must be non-empty.

Either ' or " may be used for literal quotes.

The pseudo-variable **syspnt** is not available.

SEE ALSO

awk(1).

"SNOBOL, a String Manipulation Language," by D. J. Farber, R. E. Griswold, and I. P. Polonsky, *JACM* 11 (1964), pp. 21-30.

NAME

soelim - eliminate .so's from nroff input

SYNOPSIS

soelim [file ...]

DESCRIPTION

Soelim reads the specified files or the standard input and performs the textual inclusion implied by the *nroff* directives of the form

.so somefile

when they appear at the beginning of input lines. This is useful since programs such as *tbl* do not normally do this; it allows the placement of individual tables in separate files to be run as a part of a large document.

An argument consisting of a single minus (-) is taken to be a file name corresponding to the standard input.

Note that inclusion can be suppressed by using `` instead of '.', i.e.

'so /usr/lib/tmac.s

A sample usage of *soelim* would be

soelim exum?.n | tbl | nroff -ms | col | lpr

SEE ALSO

colcrt(1), more(1)

AUTHOR

William Joy

BUGS

The format of the source commands must involve no strangeness - exactly one blank must precede and no blanks follow the file name.

1. INTRODUCTION

The purpose of this study is to investigate the effects of the proposed changes on the system.

The study was conducted in a laboratory setting. The participants were selected from a pool of volunteers.

The results of the study are presented in the following sections.

The study was conducted in a laboratory setting.

The study was conducted in a laboratory setting.

The study was conducted in a laboratory setting.

The study was conducted in a laboratory setting.

NAME

sort - sort or merge files

SYNOPSIS

```
sort [ -mubdfinrtz ] [ +pos1 [ -pos2 ] ] ... [ -o name ] [ -T direc-
tory ] [ name ] ...
```

DESCRIPTION

Sort sorts lines of all the named files together and writes the result on the standard output. The name '-' means the standard input. If no input files are named, the standard input is sorted.

The default sort key is an entire line. Default ordering is lexicographic by bytes in machine collating sequence. The ordering is affected globally by the following options, one or more of which may appear.

- b** Ignore leading blanks (spaces and tabs) in field comparisons.
- d** 'Dictionary' order: only letters, digits and blanks are significant in comparisons.
- f** Fold upper case letters onto lower case.
- i** Ignore characters outside the ASCII range 040-0176 in nonnumeric comparisons.
- n** An initial numeric string, consisting of optional blanks, optional minus sign, and zero or more digits with optional decimal point, is sorted by arithmetic value. Option **n** implies option **b**.
- r** Reverse the sense of comparisons.
- tz** 'Tab character' separating fields is **x**.

The notation **+pos1 -pos2** restricts a sort key to a field beginning at *pos1* and ending just before *pos2*. *Pos1* and *pos2* each have the form *m.n*, optionally followed by one or more of the flags **bdfinr**, where *m* tells a number of fields to skip from the beginning of the line and *n* tells a number of characters to skip further. If any flags are present they override all the global ordering options for this key. If the **b** option is in effect *n* is counted from the first nonblank in the field; **b** is attached independently to *pos2*. A missing *n* means .0; a missing **-pos2** means the end of the line. Under the **-tz** option, fields are strings separated by **x**; otherwise fields are nonempty nonblank strings separated by blanks.

When there are multiple sort keys, later keys are compared only after all earlier keys compare equal. Lines that otherwise compare equal are ordered with all bytes significant.

These option arguments are also understood:

- c** Check that the input file is sorted according to the ordering rules; give no output unless the file is out of sort.
- m** Merge only, the input files are already sorted.
- o** The next argument is the name of an output file to use instead of the standard output. This file may be the same as one of the inputs.
- T** The next argument is the name of a directory in which temporary files should be made.

- u** Suppress all but one in each set of equal lines. Ignored bytes and bytes outside keys do not participate in this comparison.

Examples. Print in alphabetical order all the unique spellings in a list of words. Capitalized words differ from uncapitalized.

```
sort -u +0f +0 list
```

Print the password file (`passwd(5)`) sorted by user id number (the 3rd colon-separated field).

```
sort -t: +2n /etc/passwd
```

Print the first instance of each month in an already sorted file of (month day) entries. The options `-um` with just one input file make the choice of a unique representative from a set of equal lines predictable.

```
sort -um +0 -1 dates
```

FILES

`/usr/tmp/stm*`, `/tmp/*`: first and second tries for temporary files

SEE ALSO

`uniq(1)`, `comm(1)`, `rev(1)`, `join(1)`

DIAGNOSTICS

Comments and exits with nonzero status for various trouble conditions and for disorder discovered under option `-c`.

BUGS

Very long lines are silently truncated.

NAME

spell, spellin, spellout — find spelling errors

SYNOPSIS

```
spell [ option ] ... [ file ] ...  
/usr/dict/spellin [ list ]  
/usr/dict/spellout [ -d ] list
```

DESCRIPTION

Spell collects words from the named documents, and looks them up in a spelling list. Words that neither occur among nor are derivable (by applying certain inflections, prefixes or suffixes) from words in the spelling list are printed on the standard output. If no files are named, words are collected from the standard input.

Spell ignores most *troff*, *tbl* and *eqn*(1) constructions.

Under the *-v* option, all words not literally in the spelling list are printed, and plausible derivations from spelling list words are indicated.

Under the *-b* option, British spelling is checked. Besides preferring *centre*, *colour*, *speciality*, *travelled*, etc., this option insists upon *-ise* in words like *standardise*. Fowler and the OED to the contrary notwithstanding.

Under the *-x* option, every plausible stem is printed with '=' for each word.

The spelling list is based on many sources, and while more haphazard than an ordinary dictionary, is also more effective in respect to proper names and popular technical words. Coverage of the specialized vocabularies of biology, medicine and chemistry is light.

Pertinent auxiliary files may be specified by name arguments, indicated below with their default settings. Copies of all output are accumulated in the history file. The stop list filters out misspellings (e.g. *thier=thy-y+ier*) that would otherwise pass.

Two routines help maintain the hash lists used by *spell*. Both expect a list of words, one per line, from the standard input. *Spellin* adds the words on the standard input to the preexisting *list* and places a new list on the standard output. If no *list* is specified, the new list is created from scratch. *Spellout* looks up each word in the standard input and prints on the standard output those that are missing from (or present on, with option *-d*) the hash list.

FILES

D=/usr/dict/hlist[ab]: hashed spelling lists, American & British
S=/usr/dict/hstop: hashed stop list
H=/usr/dict/spellhist: history file
/usr/lib/spell
deroff(1), sort(1), tee(1), sed(1)

BUGS

The spelling list's coverage is uneven; new installations will probably wish to monitor the output for several months to gather local additions. British spelling was done by an American.

NAME

spline - interpolate smooth curve

SYNOPSIS

spline [options]

DESCRIPTION

Spline takes pairs of numbers from the standard input as abscissas and ordinates of a function. It produces a similar set, which is approximately equally spaced and includes the input set, on the standard output. The cubic spline output (R. W. Hamming, *Numerical Methods for Scientists and Engineers*, 2nd ed., pp. 349ff) has two continuous derivatives, and sufficiently many points to look smooth when plotted, for example by *graph*(1G).

The following *options* are recognized, each as a separate argument:

- a Supply abscissas automatically (they are missing from the input); spacing is given by the next argument, or is assumed to be 1 if next argument is not a number.
- k The constant k used in the boundary value computation:
 $y_0'' = ky_1'$, $y_n'' = ky_{n-1}'$
is set by the next argument (default $k = 0$).
- n Space output points so that approximately n intervals occur between the lower and upper x limits (default $n = 100$).
- p Make output periodic, i.e., match derivatives at ends. First and last input values should normally agree.
- x Next 1 (or 2) arguments are lower (and upper) x limits. Normally, these limits are calculated from the data. Automatic abscissas start at lower limit (default 0).

SEE ALSO

graph(1G).

DIAGNOSTICS

When data is not strictly monotone in x , *spline* reproduces the input without interpolating extra points.

BUGS

A limit of 1,000 input points is enforced silently.

The first part of the paper is devoted to a discussion of the general principles of the theory of the structure of the atom.

In the second part of the paper the author discusses the question of the structure of the atom in the case of the hydrogen atom.

The third part of the paper is devoted to a discussion of the question of the structure of the atom in the case of the helium atom.

In the fourth part of the paper the author discusses the question of the structure of the atom in the case of the lithium atom.

The fifth part of the paper is devoted to a discussion of the question of the structure of the atom in the case of the beryllium atom.

In the sixth part of the paper the author discusses the question of the structure of the atom in the case of the boron atom.

The seventh part of the paper is devoted to a discussion of the question of the structure of the atom in the case of the carbon atom.

In the eighth part of the paper the author discusses the question of the structure of the atom in the case of the nitrogen atom.

The ninth part of the paper is devoted to a discussion of the question of the structure of the atom in the case of the oxygen atom.

NAME

split - split a file into pieces

SYNOPSIS

split [-n] [file [name]]

DESCRIPTION

Split reads *file* and writes it in *n*-line pieces (default 1000), as many as necessary, onto a set of output files. The name of the first output file is *name* with *aa* appended, and so on lexicographically. If no output name is given, *x* is default.

If no input file is given, or if *-* is given in its stead, then the standard input file is used.

1. The first part of the report is a general
description of the project and its objectives.
2. The second part is a detailed description of the
methodology used in the study.
3. The third part is a description of the results
of the study.

NAME

stctrl, *stskip* - special streamer features, skip files

SYNOPSIS

stctrl [-e] [-r]
stskip count

DESCRIPTION

Stctrl is used to execute special streamer commands. The possible options are:

-e to erase the whole tape.

-r to make a retention of the tape.

Stskip skips the next files on the tape. Count holds the number of files to be skipped.

These two commands always refer to streamer drive 0.

FILES

/dev/rst0
/dev/nrst0

SEE ALSO

st(4)

NAME

`stksiz` - set stacksize

SYNOPSIS

`stksiz size file ...`

DESCRIPTION

`Stksiz` sets the stacksize of the executable program *file* to *size* bytes.

A default size of 2k bytes is set by the linker. For larger or recursive programs you will have to increase this amount. See in the EXAMPLE section of `adb(1)` how to recognize a stack overflow.

The `stksiz` command is necessary because an automatic expansion of the stack in case of a stack overflow is in general impossible with the MC 68000, as it knows no bus-error-recovery. The system can recover from a stack overflow and increase the stack size automatically, if the executed code was e.g. "move.l xxx,-2042(a6)", or even "move.w xxx,-(a7)", but not from "move.l xxx,-(a7)", as there is no way of finding out whether the error occurred on the first or second halfword.

With the MC 68010 this command is obsolete.

SEE ALSO

`size(1)`

NAME

strings - find the printable strings in a object, or other binary, file

SYNOPSIS

strings [-] [-o] [-number] file ...

DESCRIPTION

Strings looks for ascii strings in a binary file. A string is any sequence of 4 or more printing characters ending with a newline or a null. Unless the - flag is given, *strings* only looks in the initialized data space of object files. If the -o flag is given, then each string is preceded by its offset in the file (in octal). If the -number flag is given then number is used as the minimum string length rather than 4.

Strings is useful for identifying random object files and many other things.

SEE ALSO

od(1)

BUGS

The algorithm for identifying strings is extremely primitive

1940

1941

1942

1943

1944

1945

1946

1947

NAME

`strip` - remove symbols and relocation bits

SYNOPSIS

`strip` name ...

DESCRIPTION

Strip removes the symbol table and relocation bits ordinarily attached to the output of the assembler and link editor. This is useful to save space after a program has been debugged.

The effect of *strip* is the same as use of the `-s` option of *ld*.

If *name* is an archive file, *strip* will remove the local symbols from any *a.out* format files it finds in the archive. Certain libraries, such as those residing in `/lib`, have no need for local symbols. By deleting them, the size of the archive is decreased and link editing performance is increased.

FILES

`/tmp/stm*` temporary file

SEE ALSO

`ld(1)`.

The first part of the report is a general description of the project. It includes a brief history of the project, a statement of the purpose, and a description of the scope. The second part of the report is a detailed description of the project. It includes a description of the methodology, a description of the results, and a description of the conclusions. The third part of the report is a discussion of the project. It includes a discussion of the strengths and weaknesses of the project, a discussion of the implications of the project, and a discussion of the future of the project.

NAME

struct - structure Fortran programs

SYNOPSIS

struct [option] ... file

DESCRIPTION

Struct translates the Fortran program specified by *file* (standard input default) into a Ratfor program. Wherever possible, Ratfor control constructs replace the original Fortran. Statement numbers appear only where still necessary. Cosmetic changes are made, including changing Hollerith strings into quoted strings and relational operators into symbols (e.g. '.GT.' into '>'). The output is appropriately indented.

The following options may occur in any order.

- s Input is accepted in standard format, i.e. comments are specified by a c, C, or * in column 1, and continuation lines are specified by a nonzero, nonblank character in column 6. Normally, a statement whose first nonblank character is not alphanumeric is treated as a continuation.
- i Do not turn computed goto statements into switches. (Ratfor does not turn switches back into computed goto statements.)
- a Turn sequences of else ifs into a non-Ratfor switch of the form

```
switch {  
    case pred1: code  
    case pred2: code  
    case pred3: code  
    default: code  
}
```

The case predicates are tested in order; the code appropriate to only one case is executed. This generalized form of switch statement does not occur in Ratfor.

- b Generate goto's instead of multilevel break statements.
- n Generate goto's instead of multilevel next statements.
- en If *n* is 0 (default), place code within a loop only if it can lead to an iteration of the loop. If *n* is nonzero, admit code segments with fewer than *n* statements to a loop if otherwise the loop would have exits to several places including the segment, and the segment can be reached only from the loop.

FILES

/tmp/struct*
/usr/lib/struct/*

SEE ALSO

f77(1)

BUGS

Struct knows Fortran 66 syntax, but not full Fortran 77 (alternate returns, IF...THEN...ELSE, etc.)

If an input Fortran program contains identifiers which are reserved

STRUCT(1)

MUNIX

STRUCT(1)

words in Ratfor, the structured version of the program will not be a valid Ratfor program.
Extended range DO's generate cryptic errors.
Columns 73-80 are not special even when -s is in effect.
Will not generate Ratfor FOR statements.

NAME

`stty` - set the options for a terminal

SYNOPSIS

`stty [-a] [-g] [options]`

DESCRIPTION

Stty sets certain terminal I/O options for the device that is the current standard input; without arguments, it reports the settings of certain options; with the `-a` option, it reports all of the option settings; with the `-g` option, it reports current settings in a form that can be used as an argument to another *stty* command. Detailed information about the modes listed in the first five groups below may be found in *termio*(4). Options in the last group are implemented using options in the previous groups. Note that many combinations of options make no sense, but no sanity checking is performed. The options are selected from the following:

Control Modes

`parenb (-parenb)` enable (disable) parity generation and detection.
`parodd (-parodd)` select odd (even) parity.
`cs5 cs6 cs7 cs8` select character size (see *termio*(4)).
`0` hang up phone line immediately.
`50 75 110 134 150 200 300 600 1200 1800 2400 4800 9600 exta extb`
 Set terminal baud rate to the number given, if possible. (All speeds are not supported by all hardware interfaces.)
`hupcl (-hupcl)` hang up (do not hang up) a DATA-PHONE® data set connection on last close.
`hup (-hup)` same as `hupcl (-hupcl)`.
`cstopb (-cstopb)` use two (one) stop bits per character.
`cread (-cread)` enable (disable) the receiver.
`clocal (-clocal)` assume a line without (with) modem control.

Input Modes

`ignbrk (-ignbrk)` ignore (do not ignore) break on input.
`brkint (-brkint)` signal (do not signal) INTR on break.
`ignpar (-ignpar)` ignore (do not ignore) parity errors.
`parmrk (-parmrk)` mark (do not mark) parity errors (see *termio*(4)).
`inpck (-inpck)` enable (disable) input parity checking.
`istrip (-istrip)` strip (do not strip) input characters to seven bits.
`inlcr (-inlcr)` map (do not map) NL to CR on input.
`igncr (-igncr)` ignore (do not ignore) CR on input.
`icrnl (-icrnl)` map (do not map) CR to NL on input.
`iuclic (-iuclic)` map (do not map) upper-case alphabets to lower case on input.
`ixon (-ixon)` enable (disable) START/STOP output control. Output is stopped by sending an ASCII DC3 and started by sending an ASCII DC1.
`ixany (-ixany)` allow any character (only DC1) to restart output.
`ixoff (-ixoff)` request that the system send (not send) START/STOP characters when the input queue is nearly empty/full.

Output Modes

opost (-opost)	post-process output (do not post-process output; ignore all other output modes).
olcuc (-olcuc)	map (do not map) lower-case alphabets to upper case on output.
onlcr (-onlcr)	map (do not map) NL to CR-NL on output.
ocrnl (-ocrnl)	map (do not map) CR to NL on output.
onocr (-onocr)	do not (do) output CRs at column zero.
onlret (-onlret)	on the terminal NL performs (does not perform) the CR function.
ofill (-ofill)	use fill characters (use timing) for delays.
ofdel (-ofdel)	fill characters are DELs (NULs).
cr0 cr1 cr2 cr3	select style of delay for carriage returns (see <i>termio</i> (4)).
nl0 nl1	select style of delay for line-feeds (see <i>termio</i> (4)).
tab0 tab1 tab2 tab3	select style of delay for horizontal tabs (see <i>termio</i> (4)).
bs0 bs1	select style of delay for backspaces (see <i>termio</i> (4)).
ff0 ff1	select style of delay for form-feeds (see <i>termio</i> (4)).
vt0 vt1	select style of delay for vertical tabs (see <i>termio</i> (4)).
Local Modes	
isig (-isig)	enable (disable) the checking of characters against the special control characters INTR and QUIT.
icanon (-icanon)	enable (disable) canonical input (ERASE and KILL processing).
xcase (-xcase)	canonical (unprocessed) upper/lower-case presentation.
echo (-echo)	echo back (do not echo back) every character typed.
echoe (-echoe)	echo (do not echo) ERASE character as a backspace-space-backspace string. Note: this mode will erase the ERASEd character on many CRT terminals; however, it does <i>not</i> keep track of column position and, as a result, may be confusing on escaped characters, tabs, and backspaces.
echok (-echok)	echo (do not echo) NL after KILL character.
lfkc (-lfkc)	the same as echok (-echok); obsolete.
echonl (-echonl)	echo (do not echo) NL.
noflsh (-noflsh)	disable (enable) flush after INTR or QUIT.
stwrap (-stwrap)	disable (enable) truncation of lines longer than 79 characters on a synchronous line.
stflush (-stflush)	enable (disable) flush on a synchronous line after every <i>write</i> (2).
stappl (-stappl)	use application mode (use line mode) on a synchronous line.
Control Assignments	
control-character c	set <i>control-character</i> to <i>c</i> , where <i>control-character</i> is <i>erase</i> , <i>kill</i> , <i>intr</i> , <i>quit</i> , <i>eof</i> , <i>eol</i> , <i>min</i> , or <i>time</i> (<i>min</i> and <i>time</i> are used with -icanon ; see <i>termio</i> (4)). If <i>c</i> is preceded by an (escaped from the shell) caret (^), then the value used is the corresponding CTRL character (e.g., " ^d " is a CTRL-d); " ^? " is interpreted as DEL and

"^_" is interpreted as undefined.
 set line discipline to *i* ($0 < i < 127$).

line *i*

Combination Modes

evenp or parity enable **parenb** and **cs7**.
oddp enable **parenb**, **cs7**, and **parodd**.
-parity, -evenp, or -oddp disable **parenb**, and set **cs8**.
raw (-raw or cooked) enable (disable) raw input and output (no ERASE, KILL, INTR, QUIT, EOT, or output post processing).
nl (-nl) unset (set) **icrnl**, **onlcr**. In addition **-nl** unsets **inlcr**, **igncr**, **ocrnl**, and **onlret**.
lcase (-lcase) set (unset) **xcase**, **iucLC**, and **olcuc**.
LCASE (-LCASE) same as **lcase (-lcase)**.
tabs (-tabs or tab3) preserve (expand to spaces) tabs when printing.
ek reset ERASE and KILL characters back to normal BS and CTRL-X.
sane resets all modes to some reasonable values.
term set all modes suitable for the terminal type *term*, where *term* is one of **tty33**, **tty37**, **vt05**, **tn300**, **ti700**, or **tek**.

SEE ALSO

tabs(1), **ioctl(2)**, **termio(4)**

NAME

style - analyze surface characteristics of a document

SYNOPSIS

style [*-ml*] [*-mm*] [*-a*] [*-e*] [*-l num*] [*-r num*] [*-p*] [*-P*] file ...

DESCRIPTION

Style analyzes the surface characteristics of the writing style of a document. It reports on readability, sentence length and structure, word length and usage, verb type, and sentence openers. Because *style* runs *deroff* before looking at the text, formatting header files should be included as part of the input. The default macro package *-ms* may be overridden with the flag *-mm*. The flag *-ml*, which causes *deroff* to skip lists, should be used if the document contains many lists of non-sentences. The other options are used to locate sentences with certain characteristics.

-a print all sentences with their length and readability index.

-e print all sentences that begin with an expletive.

-p print all sentences that contain a passive verb.

-l num
print all sentences longer than *num*.

-r num
print all sentences whose readability index is greater than *num*.

-P print parts of speech of the words in the document.

SEE ALSO

deroff(1), *diction*(1)

BUGS

Use of non-standard formatting macros may cause incorrect sentence breaks.

NAME

`su` - become super-user or another user

SYNOPSIS

`su [-] [name [arg ...]]`

DESCRIPTION

`Su` allows one to become another user without logging off. The default user *name* is *root* (i.e., super-user).

To use `su`, the appropriate password must be supplied (unless one is already super-user). If the password is correct, `su` will execute a new shell with the user ID set to that of the specified user. To restore normal user ID privileges, type an EOF to the new shell.

Any additional arguments are passed to the shell, permitting the super-user to run shell procedures with restricted privileges (an *arg* of the form `-c string` executes *string* via the shell). When additional arguments are passed, `/bin/sh` is always used. When no additional arguments are passed, `su` uses the shell specified in the password file.

An initial `-` flag causes the environment to be changed to the one that would be expected if the user actually logged in again. This is done by invoking the shell with an *arg0* of `-su` causing the *.profile* in the home directory of the new user ID to be executed. Note that the *.profile* can check *arg0* for `-sh` or `-su` to determine how it was invoked.

EXAMPLES

```
su
su - adm -c "/usr/lib/acct/ckpacct"
```

FILES

`/etc/passwd` system's password file
`$HOME/.profile` user's profile

SEE ALSO

`env(1)`, `login(1)`, `sh(1)`, `environ(7)`.

NAME

sum - sum and count blocks in a file

SYNOPSIS

sum file

DESCRIPTION

Sum calculates and prints a 16-bit checksum for the named file, and also prints the number of blocks in the file. It is typically used to look for bad spots, or to validate a file communicated over some transmission line.

SEE ALSO

wc(1)

DIAGNOSTICS

'Read error' is indistinuishable from end of file on most devices; check the block count.

NAME

tabs - set tabs on a terminal

SYNOPSIS

tabs [tabspec] [+mn] [-Ttype]

DESCRIPTION

Tabs sets the tab stops on the user's terminal according to the tab specification *tabspec*, after clearing any previous settings. The user must of course be logged in on a terminal with remotely-settable hardware tabs.

Users of GE TermiNet terminals should be aware that they behave in a different way than most other terminals for some tab settings: the first number in a list of tab settings becomes the *left margin* on a TermiNet terminal. Thus, any list of tab numbers whose first element is other than 1 causes a margin to be left on a TermiNet, but not on other terminals. A tab list beginning with 1 causes the same effect regardless of terminal type. It is possible to set a left margin on some other terminals, although in a different way (see below).

Four types of tab specification are accepted for *tabspec*: "canned," repetitive, arbitrary, and file. If no *tabspec* is given, the default value is -8, i.e., UNIX System "standard" tabs. The lowest column number is 1. Note that for *tabs*, column 1 always refers to the leftmost column on a terminal, even one whose column markers begin at 0, e.g., the DASI 300, DASI 300s, and DASI 450.

-code Gives the name of one of a set of "canned" tabs. The legal codes and their meanings are as follows:

-a 1,10,16,36,72

Assembler, IBM S/370, first format

-a2 1,10,16,40,72

Assembler, IBM S/370, second format

-c 1,8,12,16,20,55

COBOL, normal format

-c2 1,6,10,14,49

COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a tab reaches column 12. Files using this tab setup should include a format specification as follows:

<:t-c2 m6 s66 d:>

-c3 1,6,10,14,18,22,26,30,34,38,42,46,50,54,58,62,67

COBOL compact format (columns 1-6 omitted), with more tabs than -c2. This is the recommended format for COBOL. The appropriate format specification is:

<:t-c3 m6 s66 d:>

-f 1,7,11,15,19,23

FORTRAN

-p 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61

PL/I

-s 1,10,55

SNOBOL

-u 1,12,20,44
UNIVAC 1100 Assembler

In addition to these "canned" formats, three other types exist:

-n A repetitive specification requests tabs at columns $1+n$, $1+2n$, etc. Note that such a setting leaves a left margin of n columns on TerminoNet terminals *only*. Of particular importance is the value **-8**: this represents the UNIX System "standard" tab setting, and is the most likely tab setting to be found at a terminal. It is required for use with the *nroff* **-h** option for high-speed output. Another special case is the value **-0**, implying no tabs at all.

n1,n2,...

The arbitrary format permits the user to type any chosen set of numbers, separated by commas, in ascending order. Up to 40 numbers are allowed. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the tab lists 1,10,20,30 and 1,10,+10,+10 are considered identical.

—file If the name of a file is given, *tabs* reads the first line of the file, searching for a format specification. If it finds one there, it sets the tab stops according to it, otherwise it sets them as **-8**. This type of specification may be used to make sure that a tabbed file is printed with correct tab settings, and would be used with the *pr*(1) command:

tabs -- file; pr file

Any of the following may be used also; if a given flag occurs more than once, the last value given takes effect:

-Ttype

Tabs usually needs to know the type of terminal in order to set tabs and always needs to know the type to set margins. *Type* is a name listed in *term*(5). If no **-T** flag is supplied, *tabs* searches for the **\$TERM** value in the *environment* (see *environ*(5)). If no *type* can be found, *tabs* tries a sequence that will work for many terminals.

+mn The margin argument may be used for some terminals. It causes all tabs to be moved over n columns by making column $n+1$ the left margin. If **+m** is given without a value of n , the value assumed is 10. For a TerminoNet, the first value in the tab list should be 1, or the margin will move even further to the right. The normal (leftmost) margin on most terminals is obtained by **+m0**. The margin for most terminals is reset only when the **+m** flag is given explicitly.

Tab and margin setting is performed via the standard output.

DIAGNOSTICS

illegal tabs

when arbitrary tabs are ordered incorrectly.

illegal increment

when a zero or missing increment is found in an arbitrary specification.

unknown tab code

when a "canned" code cannot be found.

can't open

if **—file** option used, and file can't be opened.

file indirection

if *—file* option used and the specification in that file points to yet another file. Indirection of this form is not permitted.

SEE ALSO

nroff(1), environ(7), term(7).

BUGS

There is no consistency among different terminals regarding ways of clearing tabs and setting the left margin.

It is generally impossible to usefully change the left margin without also setting tabs.

Tabs clears only 20 tabs (on terminals requiring a long sequence), but is willing to set 40.

NAME

tail - deliver the last part of a file

SYNOPSIS

tail [±number[lbc]] [file]

DESCRIPTION

Tail copies the named file to the standard output beginning at a designated place. If no file is named, the standard input is used.

Copying begins at distance *+number* from the beginning, or *-number* from the end of the input. *Number* is counted in units of lines, blocks or characters, according to the appended option l, b or c. When no units are specified, counting is by lines.

SEE ALSO

dd(1)

BUGS

Tails relative to the end of the file are treasured up in a buffer, and thus are limited in length. Various kinds of anomalous behavior may happen with character special files.

... of the ...
... of the ...
... of the ...

... of the ...
... of the ...
... of the ...

NAME

tar - tape archiver

SYNOPSIS

tar [key] [name ...]

DESCRIPTION

Tar saves and restores files on magtape. Its actions are controlled by the *key* argument. The *key* is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped or restored. In all cases, appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

The function portion of the key is specified by one of the following letters:

- r** The named files are written on the end of the tape. The **c** function implies this.
- x** The named files are extracted from the tape. If the named file matches a directory whose contents had been written onto the tape, this directory is (recursively) extracted. The owner, modification time, and mode are restored (if possible). If no file argument is given, the entire content of the tape is extracted. Note that if multiple entries specifying the same file are on the tape, the last one overwrites all earlier.
- t** The names of the specified files are listed each time they occur on the tape. If no file argument is given, all of the names on the tape are listed.
- u** The named files are added to the tape if either they are not already there or have been modified since last put on the tape.
- c** Create a new tape; writing begins on the beginning of the tape instead of after the last file. This command implies **r**.

The following characters may be used in addition to the letter which selects the function desired.

- 0,...,7** This modifier selects the drive on which the tape is mounted. The default is 1.
- v** Normally *tar* does its work silently. The **v** (verbose) option causes it to type the name of each file it treats preceded by the function letter. With the **t** function, **v** gives more information about the tape entries than just the name.
- w** causes *tar* to print the action to be taken followed by file name, then wait for user confirmation. If a word beginning with 'y' is given, the action is performed. Any other input means don't do it.
- f** causes *tar* to use the next argument as the name of the archive instead of /dev/mt?. If the name of the file is '-', *tar* writes to standard output or reads from standard input, whichever is appropriate. Thus, *tar* can be used as the head or tail of a filter chain. *Tar* can also be used to move hierarchies with the

command

cd fromdir; tar cf - . | (cd todir; tar xf -)

- b** causes *tar* to use the next argument as the blocking factor for tape records. The default is 1, the maximum is 127. This option should only be used with raw magnetic tape archives (See *f* above). The block size is determined automatically when reading tapes (key letters 'x' and 't').
- l** tells *tar* to complain if it cannot resolve all of the links to the files dumped. If this is not specified, no error messages are printed.
- m** tells *tar* to not restore the modification times. The mod time will be the time of extraction.

FILES

/dev/mt?
/dev/rmt?
/tmp/tar*

DIAGNOSTICS

Complaints about bad key characters and tape read/write errors.
Complaints if enough memory is not available to hold the link tables.

BUGS

There is no way to ask for the *n*-th occurrence of a file.
Tape errors are handled ungracefully.
The *u* option can be slow.
The *b* option should not be used with archives that are going to be updated. If the archive is on a disk file the *b* option should not be used at all, as updating an archive stored in this manner can destroy it.
The current limit on file name length is 100 characters.

NAME

`tbl` - format tables for `nroff` or `troff`

SYNOPSIS

`tbl [-TX] [files]`

DESCRIPTION

`Tbl` is a preprocessor that formats tables for `nroff`(1) or `troff`(1). The input files are copied to the standard output, except for lines between `.TS` and `.TE` command lines, which are assumed to describe tables and are re-formatted by `tbl`. (The `.TS` and `.TE` command lines are not altered by `tbl`).

`.TS` is followed by global options. The available global options are:

<code>center</code>	center the table (default is left-adjust);
<code>expand</code>	make the table as wide as the current line length;
<code>box</code>	enclose the table in a box;
<code>doublebox</code>	enclose the table in a double box;
<code>allbox</code>	enclose each item of the table in a box;
<code>tab(x)</code>	use the character <code>x</code> instead of a tab to separate items in a line of input data.

The global options, if any, are terminated with a semi-colon (;).

Next come lines describing the format of each line of the table. Each such format line describes one line of the actual table, except that the last format line (which must end with a period) describes *all* remaining lines of the table. Each column of each line of the table is described by a single key-letter, optionally followed by specifiers that determine the font and point size of the corresponding item, that indicate where vertical bars are to appear between columns, that determine column width, inter-column spacing, etc. The available key-letters are:

<code>c</code>	center item within the column;
<code>r</code>	right-adjust item within the column;
<code>l</code>	left-adjust item within the column;
<code>n</code>	numerically adjust item in the column: units positions of numbers are aligned vertically;
<code>s</code>	span previous item on the left into this column;
<code>a</code>	center longest line in this column and then left-adjust all other lines in this column with respect to that centered line;
<code>^</code>	span down previous entry in this column;
<code>_</code>	replace this entry with a horizontal line;
<code>=</code>	replace this entry with a double horizontal line.

The characters `B` and `I` stand for the bold and italic fonts, respectively; the character `|` indicates a vertical line between columns.

The format lines are followed by lines containing the actual data for the table, followed finally by `.TE`. Within such data lines, data items are normally separated by tab characters.

If a data line consists of only `_` or `=`, a single or double line, respectively, is drawn across the table at that point; if a *single item* in a data line consists of only `_` or `=`, then that item is replaced by a single or double line.

Full details of all these and other features of *tbl* are given in the reference manual cited below.

The *-TX* option forces *tbl* to use only full vertical line motions, making the output more suitable for devices that cannot generate partial vertical line motions (e.g., line printers).

If no file names are given as arguments, *tbl* reads the standard input, so it may be used as a filter. When it is used with *eqn(1)* or *neqn(1)*, *tbl* should come first to minimize the volume of data passed through pipes.

EXAMPLE

If we let \rightarrow represent a tab (which should be typed as a genuine tab), then the input:

```
.TS
center box ;
cB s s
cl | cl s
^ | c c
l | n n .
Household Population
=
Town→Households
→Number→Size
=
Bedminster→789→3.26
Bernards Twp.→3087→3.74
Bernardsville→2018→3.30
Bound Brook→3425→3.04
Bridgewater→7897→3.81
Far Hills→240→3.19
.TE
```

yields:

Household Population		
Town	Households	
	Number	Size
Bedminster	789	3.26
Bernards Twp.	3087	3.74
Bernardsville	2018	3.30
Bound Brook	3425	3.04
Bridgewater	7897	3.81
Far Hills	240	3.19

SEE ALSO

TBL - A Program to Format Tables by M. E. Lesk
eqn(1), *mm(1)*, *mmt(1)*, *troff(1)*, *mm(7)*, *mv(7)*.

BUGS

See *BUGS* under *troff(1)*.

NAME

`tc` - phototypesetter simulator

SYNOPSIS

`tc [-t] [-sn] [-pl] [file]`

DESCRIPTION

`Tc` interprets its input (standard input default) as device codes for a Wang Laboratories, Inc. C/A/T phototypesetter. The standard output of `tc` is intended for a Tektronix 4014 terminal with ASCII and APL character sets. The sixteen typesetter sizes are mapped into the 4014's four sizes; the entire TROFF character set is drawn using the 4014's character generator, with overstruck combinations where necessary. Typical usage is:

`troff -t files | tc`

At the end of each page, `tc` waits for a new-line (empty line) from the keyboard before continuing on to the next page. In this wait state, the command `e` will *suppress* the screen erase before the next page; `sn` will cause the next `n` pages to be skipped; and `!cmd` will send `cmd` to the shell.

The command line options are:

- `-t` Don't wait between pages (for directing output into a file).
- `-sn` Skip the first `n` pages.
- `-pl` Set page length to `l`; `l` may include the scale factors `p` (points), `i` (inches), `c` (centimeters), and `P` (picas); default is picas.

SEE ALSO

`4014(1)`, `sh(1)`, `tplot(1G)`, `troff(1)`.

BUGS

Font distinctions are lost.

NAME

tee - pipe fitting

SYNOPSIS

tee [-i] [-a] [file] ...

DESCRIPTION

Tee transcribes the standard input to the standard output and makes copies in the *files*. Option *-i* ignores interrupts; option *-a* causes the output to be appended to the *files* rather than overwriting them.

NAME

test - condition evaluation command

SYNOPSIS

```
test expr
[ expr ]
```

DESCRIPTION

Test evaluates the expression *expr* and, if its value is true, returns a zero (true) exit status; otherwise, a non-zero (false) exit status is returned; *test* also returns a non-zero exit status if there are no arguments. The following primitives are used to construct *expr*:

- r *file* true if *file* exists and is readable.
- w *file* true if *file* exists and is writable.
- x *file* true if *file* exists and is executable.
- f *file* true if *file* exists and is a regular file.
- d *file* true if *file* exists and is a directory.
- c *file* true if *file* exists and is a character special file.
- b *file* true if *file* exists and is a block special file.
- p *file* true if *file* exists and is a named pipe (fifo).
- u *file* true if *file* exists and its set-user-ID bit is set.
- g *file* true if *file* exists and its set-group-ID bit is set.
- k *file* true if *file* exists and its sticky bit is set.
- s *file* true if *file* exists and has a size greater than zero.
- t [*fdes*] true if the open file whose file descriptor number is *fdes* (1 by default) is associated with a terminal device.
- z *s1* true if the length of string *s1* is zero.
- n *s1* true if the length of the string *s1* is non-zero.
- s1* = *s2* true if strings *s1* and *s2* are identical.
- s1* != *s2* true if strings *s1* and *s2* are *not* identical.
- s1* true if *s1* is *not* the null string.
- n1* -eq *n2* true if the integers *n1* and *n2* are algebraically equal. Any of the comparisons -ne, -gt, -ge, -lt, and -le may be used in place of -eq.

These primaries may be combined with the following operators:

- ! unary negation operator.
- a binary *and* operator.
- o binary *or* operator (-a has higher precedence than -o).
- (*expr*) parentheses for grouping.

Notice that all the operators and flags are separate arguments to *test*. Notice also that parentheses are meaningful to the shell and, therefore, must be escaped.

SEE ALSO

find(1), sh(1).

WARNING

In the second form of the command (i.e., the one that uses `[]`, rather than the word *test*), the square brackets must be delimited by blanks. Some UNIX systems do not recognize the second form of the command.

NAME

time - time a command

SYNOPSIS

time command

DESCRIPTION

The given command is executed; after it is complete, *time* prints the elapsed time during the command, the time spent in the system, and the time spent in execution of the command. Times are reported in seconds.

The times are printed on the diagnostic output stream.

BUGS

Elapsed time is accurate to the second, while the CPU times are measured to the 50th or 60th second, dependent on the line frequency and the definition of HZ in /usr/sys/conf.h. Thus the sum of the CPU times can be up to a second larger than the elapsed time.

1. The first part of the report is a general description of the project and its objectives. It includes a brief history of the project and a statement of the problem to be solved. The second part is a description of the methods used in the study. This includes a description of the data collection methods and the statistical methods used to analyze the data. The third part is a description of the results of the study. This includes a description of the data and a discussion of the findings. The fourth part is a conclusion and a list of references.

2. The first part of the report is a general description of the project and its objectives. It includes a brief history of the project and a statement of the problem to be solved. The second part is a description of the methods used in the study. This includes a description of the data collection methods and the statistical methods used to analyze the data. The third part is a description of the results of the study. This includes a description of the data and a discussion of the findings. The fourth part is a conclusion and a list of references.

NAME

timex - time a command; report process data and system activity

SYNOPSIS

timex [options] command

DESCRIPTION

The given *command* is executed; the elapsed time, user time and system time spent in execution are reported in seconds. Optionally, process accounting data for the *command* and all its children can be listed or summarized, and total system activity during the execution interval can be reported.

The output of *timex* is written on standard error.

Options are:

- p List process accounting records for *command* and all its children. Suboptions *f*, *h*, *k*, *m*, *r*, and *t* modify the data items reported, as defined in *acctcom*(1). The number of blocks read or written and the number of characters transferred are always reported.
- o Report the total number of blocks read or written and total characters transferred by *command* and all its children.
- s Report total system activity (not just that due to *command*) that occurred during the execution interval of *command*. All the data items listed in *sar*(1) are reported.

SEE ALSO

acctcom(1), sar(1).

WARNING

Process records associated with *command* are selected from the accounting file */usr/adm/pacct* by inference, since process genealogy is not available. Background processes having the same user-id, terminal-id, and execution time window will be spuriously included.

EXAMPLES

A simple example:

```
timex -ops sleep 60
```

A terminal session of arbitrary complexity can be measured by timing a sub-shell:

```
timex -opskmt sh
      session commands
EOT
```

1. The first part of the report deals with the general situation of the country and the progress of the work during the year. It is a summary of the work done and the results obtained. It is a general statement of the work done and the results obtained.

2. The second part of the report deals with the details of the work done during the year. It is a detailed statement of the work done and the results obtained. It is a detailed statement of the work done and the results obtained.

3. The third part of the report deals with the financial statement of the work done during the year. It is a statement of the financial statement of the work done and the results obtained. It is a statement of the financial statement of the work done and the results obtained.

4. The fourth part of the report deals with the conclusions of the work done during the year. It is a statement of the conclusions of the work done and the results obtained. It is a statement of the conclusions of the work done and the results obtained.

5. The fifth part of the report deals with the recommendations of the work done during the year. It is a statement of the recommendations of the work done and the results obtained. It is a statement of the recommendations of the work done and the results obtained.

NAME

`tk` - paginator for the Tektronix 4014

SYNOPSIS

`tk [-t] [-N] [-pL] [file]`

DESCRIPTION

The output of `tk` is intended for a Tektronix 4014 terminal. `Tk` arranges for 66 lines to fit on the screen, divides the screen into *N* columns, and contributes an eight space page offset in the (default) single-column case. Tabs, spaces, and backspaces are collected and plotted when necessary. Teletype Model 37 half- and reverse-line sequences are interpreted and plotted. At the end of each page `tk` waits for a newline (empty line) from the keyboard before continuing on to the next page. In this wait state, the command `!command` will send the *command* to the shell.

The command line options are:

- `-t` Don't wait between pages; for directing output into a file.
- `-N` Divide the screen into *N* columns and wait after the last column.
- `-pL` Set page length to *L* lines.

SEE ALSO

`pr(1)`

NAME

touch - update date last modified of a file

SYNOPSIS

touch [-c] file ...

DESCRIPTION

Touch attempts to set the modified date of each *file*. This is done by reading a character from the file and writing it back.

If a *file* does not exist, an attempt will be made to create it unless the -c option is specified.

NAME

tplot - graphics filters

SYNOPSIS

tplot [-Tterminal [-e raster]]

DESCRIPTION

These commands read plotting instructions (see *plot(5)*) from the standard input and in general produce, on the standard output, plotting instructions suitable for a particular *terminal*. If no *terminal* is specified, the environment parameter **\$TERM** (see *environ(7)*) is used. Known *terminals* are:

300 DASI 300.

300S DASI 300s.

450 DASI 450.

4014 Tektronix 4014.

ver Versatec D1200A. This version of *plot* places a scan-converted image in **/usr/tmp/raster\$\$** and sends the result directly to the plotter device, rather than to the standard output. The **-e** option causes a previously scan-converted file *raster* to be sent to the plotter.

FILES

/usr/lib/t300

/usr/lib/t300s

/usr/lib/t450

/usr/lib/t4014

/usr/lib/vplot

/usr/tmp/raster\$\$

SEE ALSO

plot(3X), *plot(5)*, *term(7)*.

1947
The following is a list of the names of the persons who were present at the meeting of the Board of Directors of the United States Steel Corporation, held on the 12th day of January, 1947, at the offices of the Corporation, in the City of Pittsburgh, Pennsylvania.

Mr. J. Edgar Hoover, Chairman of the Board of Directors, presided at the meeting. Mr. J. Edgar Hoover, Chairman of the Board of Directors, presided at the meeting.

Mr. J. Edgar Hoover, Chairman of the Board of Directors, presided at the meeting. Mr. J. Edgar Hoover, Chairman of the Board of Directors, presided at the meeting.

Mr. J. Edgar Hoover, Chairman of the Board of Directors, presided at the meeting. Mr. J. Edgar Hoover, Chairman of the Board of Directors, presided at the meeting.

Mr. J. Edgar Hoover, Chairman of the Board of Directors, presided at the meeting. Mr. J. Edgar Hoover, Chairman of the Board of Directors, presided at the meeting.

Mr. J. Edgar Hoover, Chairman of the Board of Directors, presided at the meeting. Mr. J. Edgar Hoover, Chairman of the Board of Directors, presided at the meeting.

NAME

tr - translate characters

SYNOPSIS

tr [-cds] [string1 [string2]]

DESCRIPTION

Tr copies the standard input to the standard output with substitution or deletion of selected characters. Input characters found in *string1* are mapped into the corresponding characters of *string2*. When *string2* is short it is padded to the length of *string1* by duplicating its last character. Any combination of the options *-cds* may be used: *-c* complements the set of characters in *string1* with respect to the universe of characters whose ASCII codes are 01 through 0377 octal; *-d* deletes all input characters in *string1*; *-s* squeezes all strings of repeated output characters that are in *string2* to single characters.

In either string the notation *a-b* means a range of characters from *a* to *b* in increasing ASCII order. The character ** followed by 1, 2 or 3 octal digits stands for the character whose ASCII code is given by those digits. A ** followed by any other character stands for that character.

The following example creates a list of all the words in 'file1' one per line in 'file2', where a word is taken to be a maximal string of alphabets. The second string is quoted to protect ** from the Shell. 012 is the ASCII code for newline.

```
tr -cs A-Za-z '\012' <file1 >file2
```

SEE ALSO

ed(1), ascii(7)

BUGS

Won't handle ASCII NUL in *string1* or *string2*; always deletes NUL from input.

...the ... of ...

...the ... of ...

...the ... of ...

...the ... of ...

...the ... of ...

...the ... of ...

...the ... of ...

...the ... of ...

NAME

nroff, *nroff* - typeset or format text

SYNOPSIS

nroff [options] [files]

troff [options] [files]

DESCRIPTION

Nroff formats text contained in *files* (standard input by default) for printing on typewriter-like devices and line printers; similarly, *troff* formats text for a Wang Laboratories, Inc., C/A/T phototypesetter. Their capabilities are described in the *NROFF/TROFF User's Manual* cited below.

An argument consisting of a minus (-) is taken to be a file name corresponding to the standard input. The *options*, which may appear in any order, but must appear before the *files*, are:

- olist Print only pages whose page numbers appear in the *list* of numbers and ranges, separated by commas. A range *N-M* means pages *N* through *M*; an initial *-N* means from the beginning to page *N*; and a final *N-* means from *N* to the end. (See *BUGS* below.)
- n*N* Number first generated page *N*.
- s*N* Stop every *N* pages. *Nroff* will halt *after* every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a line-feed or new-line (new-lines do not work in pipelines, e.g., with *mm*(1)). This option does not work if the output of *nroff* is piped through *col*(1). *Troff* will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed. When *nroff* (*troff*) halts between pages, an ASCII BEL (in *troff*, the message *page stop*) is sent to the terminal.
- ra*N* Set register *a* (which must have a one-character name) to *N*.
- i Read standard input after *files* are exhausted.
- q Invoke the simultaneous input-output mode of the *.rd* request.
- z Print only messages generated by *.tm* (terminal message) requests.
- m*name* Prepend to the input *files* the non-compacted (ASCII text) macro file */usr/lib/tmac/tmac.name*.
- c*name* Prepend to the input *files* the compacted macro files */usr/lib/macros/cmp.[nt].[dt].name* and */usr/lib/macros/ucmp.[nt].name*.
- k*name* Compact the macros used in this invocation of *nroff*/*troff*, placing the output in files *[dt].name* in the current directory (see the May 1979 Addendum to the *NROFF/TROFF User's Manual* for details of compacting macro files).

Nroff only:

- T*name* Prepare output for specified terminal. Known *names* are 37 for the (default) TELETYPE® Model 37 terminal, *tn300* for the GE TermiNet 300 (or any terminal without half-line capability), *300s* for the DASI 300s, *300* for the DASI 300, *450* for the DASI 450, *lp* for a (generic) ASCII line printer, *382* for the DTC-382.

- 4000A for the Trendata 4000A, 832 for the Anderson Jacobson 832, X for a (generic) EBCDIC printer, and 2631 for the Hewlett Packard 2631 line printer.
- e Produce equally-spaced words in adjusted lines, using the full resolution of the particular terminal.
 - h Use output tabs during horizontal spacing to speed output and reduce output character count. Tab settings are assumed to be every 8 nominal character widths.
 - un Set the emboldening factor (number of character overstrikes) for the third font position (bold) to *n*, or to zero if *n* is missing.

Troff only:

- t Direct output to the standard output instead of the phototypesetter.
- f Refrain from feeding out paper and stopping phototypesetter at the end of the run.
- w Wait until phototypesetter is available, if it is currently busy.
- b Report whether the phototypesetter is busy or available. No text processing is done.
- a Send a printable ASCII approximation of the results to the standard output.
- p*N* Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.
- g Prepare output for the Murray Hill Computation Center phototypesetter and direct it to the standard output (see *gcat(1C)*). This option is not compatible with the *-s* option; furthermore, when this option is invoked, all *.fp* (font position) requests (if any) in the *troff* input must come before the first break, and no *.tl* requests may come before the first break.
- T*name* Use font-width tables for device *name* (the font tables are found in */usr/lib/font/name/**). Currently, no *names* are supported.

FILES

<i>/usr/lib/suftab</i>	suffix hyphenation tables
<i>/tmp/ta\$#</i>	temporary file
<i>/usr/lib/tmac/tmac.*</i>	standard macro files and pointers
<i>/usr/lib/macros/*</i>	standard macro files
<i>/usr/lib/term/*</i>	terminal driving tables for <i>nroff</i>
<i>/usr/lib/font/*</i>	font width tables for <i>troff</i>

SEE ALSO

NROFF/TROFF User's Manual by J. F. Ossanna.
A TROFF Tutorial by B. W. Kernighan.
eqn(1), *tbl(1)*, *mm(7)*.
col(1), *greek(1)*, *mm(1)* (*nroff* only).
gcat(1C), *mmt(1)*, *tc(1)*, *mv(7)* (*troff* only).

BUGS

Nroff/troff believes in Eastern Standard Time; as a result, depending on the time of the year and on your local time zone, the date that *nroff/troff* generates may be off by one day from your idea of what the date is.

When *nroff*/*troff* is used with the *-olist* option inside a pipeline (e.g., with one or more of *cw(1)*, *eqn(1)*, and *tbl(1)*), it may cause a harmless "broken pipe" diagnostic if the last page of the document is not specified in *list*.

NAME

true, *false* — provide truth values

SYNOPSIS

true

false

DESCRIPTION

True does nothing, successfully. *False* does nothing, unsuccessfully. They are typically used in input to *sh*(1) such as:

```
while true
do
    command
done
```

SEE ALSO

sh(1)

DIAGNOSTICS

True has exit status zero, *false* nonzero.

NAME

tsort - topological sort

SYNOPSIS

tsort [file]

DESCRIPTION

Tsort produces on the standard output a totally ordered list of items consistent with a partial ordering of items mentioned in the input *file*. If no *file* is specified, the standard input is understood.

The input consists of pairs of items (nonempty strings) separated by blanks. Pairs of different items indicate ordering. Pairs of identical items indicate presence, but not ordering.

SEE ALSO

lorder(1)

DIAGNOSTICS

Odd data: there is an odd number of fields in the input file.

BUGS

Uses a quadratic algorithm; not worth fixing for the typical use of ordering a library archive file.

TTY(1)

MUNIX

TTY(1)

NAME

tty - get terminal name

SYNOPSIS

tty

DESCRIPTION

Tty prints the pathname of the user's terminal.

DIAGNOSTICS

'not a tty' if the standard input file is not a terminal.

NAME

`ul` - do underlining

SYNOPSIS

`ul [-i] [-t terminal] [name ...]`

DESCRIPTION

`ul` reads the named files (or standard input if none are given) and translates occurrences of underscores to the sequence which indicates underlining for the terminal in use, as specified by the environment variable `TERM`. The `-t` option overrides the terminal kind specified in the environment. The file `/etc/termcap` is read to determine the appropriate sequences for underlining. If the terminal is incapable of underlining, but is capable of a standout mode then that is used instead. If the terminal can overstrike, or handles underlining automatically, `ul` degenerates to `cat(1)`. If the terminal cannot underline, underlining is ignored.

The `-i` option causes `ul` to indicate underlining onto by a separate line containing appropriate dashes '-'; this is useful when you want to look at the underlining which is present in an `nroff` output stream on a crt-terminal.

SEE ALSO

`man(1)`, `nroff(1)`, `colcrt(1)`

AUTHOR

Mark Horton

BUGS

`Nroff` usually outputs a series of backspaces and underlines intermixed with the text to indicate underlining. No attempt is made to optimize the backward motion.

1. The first part of the report deals with the general situation of the country and the progress of the work during the year. It is a summary of the work done and the results obtained. It is a general statement of the work done and the results obtained. It is a general statement of the work done and the results obtained.

2. The second part of the report deals with the details of the work done during the year. It is a detailed statement of the work done and the results obtained. It is a detailed statement of the work done and the results obtained.

NAME

`umask` - set file-creation mode mask

SYNOPSIS

`umask` [*ooo*]

DESCRIPTION

The user file-creation mode mask is set to *ooo*. The three octal digits refer to read/write/execute permissions for *owner*, *group*, and *others*, respectively (see `chmod(2)` and `umask(2)`). The value of each specified digit is subtracted from the corresponding "digit" specified by the system for the creation of a file (see `creat(2)`). For example, `umask 022` removes *group* and *others* write permission (files normally created with mode `777` become mode `755`; files created with mode `666` become mode `644`).

If *ooo* is omitted, the current value of the mask is printed.

Umask is recognized and executed by the shell.

SEE ALSO

`chmod(1)`, `sh(1)`, `chmod(2)`, `creat(2)`, `umask(2)`.

1940

1941

1942

1943

1944

1945

1946

NAME

`uname` - print name of current UNIX

SYNOPSIS

`uname [-snrva]`

DESCRIPTION

Uname prints the current system name of UNIX on the standard output file. It is mainly useful to determine what system one is using. The options cause selected information returned by *uname*(2) to be printed:

- `-s` print the system name (default).
- `-n` print the nodename (the nodename may be a name that the system is known by to a communications network).
- `-r` print the operating system release.
- `-v` print the operating system version.
- `-a` print all the above information.

SEE ALSO

uname(2).

THE HISTORY OF THE

1000

THE HISTORY OF THE

THE HISTORY OF THE

THE HISTORY OF THE

THE HISTORY OF THE

THE HISTORY OF THE

NAME

`unget` - undo a previous `get` of an SCCS file

SYNOPSIS

`unget [-rSID] [-s] [-n] files`

DESCRIPTION

`Unget` undoes the effect of a `get -e` done prior to creating the intended new delta. If a directory is named, `unget` behaves as though each file in the directory were specified as a named file, except that non-SCCS files and unreadable files are silently ignored. If a name of `-` is given, the standard input is read with each line being taken as the name of an SCCS file to be processed.

Keyletter arguments apply independently to each named file.

-rSID Uniquely identifies which delta is no longer intended. (This would have been specified by `get` as the "new delta"). The use of this keyletter is necessary only if two or more outstanding `gets` for editing on the same SCCS file were done by the same person (login name). A diagnostic results if the specified `SID` is ambiguous, or if it is necessary and omitted on the command line.

-s Suppresses the printout, on the standard output, of the intended delta's `SID`.

-n Causes the retention of the gotten file which would normally be removed from the current directory.

SEE ALSO

`delta(1)`, `get(1)`, `sact(1)`.

DIAGNOSTICS

Use `help(1)` for explanations.

174

174

174

174

174

174

174

174

174

174

174

174

NAME

unify relational data base system

SYNOPSIS

unify

DESCRIPTION

UNIFY is a relational data base system. The UNIFY-system is a collection of about 20 different programs. The program unify allows you to login to the UNIFY system and use the other components via menus.

SEE ALSO

UNIFY - Reference Manual Version 3.0

UNIFY - Tutorial Manual Version 3.0

both copy right UNIFY Corp.

BUGS

UNIFY is known to be relatively stable. However in some applications UNIFY-components may run out of stack and produce a core dump. In this case enlarge the stacksize (see stksiz(1)) of the program and try again.

10000

10000

10000

10000

10000

10000

10000

10000

NAME

uniq - report repeated lines in a file

SYNOPSIS

uniq [-udc [+n] [-n]] [input [output]]

DESCRIPTION

Uniq reads the input file comparing adjacent lines. In the normal case, the second and succeeding copies of repeated lines are removed; the remainder is written on the output file. Note that repeated lines must be adjacent in order to be found; see *sort*(1). If the *-u* flag is used, just the lines that are not repeated in the original file are output. The *-d* option specifies that one copy of just the repeated lines is to be written. The normal mode output is the union of the *-u* and *-d* mode outputs.

The *-c* option supersedes *-u* and *-d* and generates an output report in default style but with each line preceded by a count of the number of times it occurred.

The *n* arguments specify skipping an initial portion of each line in the comparison:

- n* The first *n* fields together with any blanks before each are ignored. A field is defined as a string of non-space, non-tab characters separated by tabs and spaces from its neighbors.
- +n* The first *n* characters are ignored. Fields are skipped before characters.

SEE ALSO

sort(1), *comm*(1)

NAME

units - conversion program

SYNOPSIS

units

DESCRIPTION

Units converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

You have: inch

You want: cm

** 2.54000e+00*

/ 3.93701e-01

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

You have: 15 pounds force/in2

You want: atm

** 1.02069e+00*

/ 9.79730e-01

Units only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi ratio of circumference to diameter

c speed of light

e charge on an electron

g acceleration of gravity

force same as g

mole Avogadro's number

water pressure head per unit height of water

au astronomical unit

'Pound' is a unit of mass. Compound names are run together, e.g. 'light-year'. British units that differ from their US counterparts are prefixed thus: 'brgallon'. Currency is denoted 'belgiumfranc', 'britainpound', ...

For a complete list of units, 'cat /usr/lib/units'.

FILES

/usr/lib/units

BUGS

Don't base your financial plans on the currency conversions.

THE UNIVERSITY OF CHICAGO
DIVISION OF THE PHYSICAL SCIENCES

REPORT OF THE
COMMISSIONER OF THE GENERAL LAND OFFICE

FOR THE YEAR 1900
AND THE PROGRESS OF THE
LAND OFFICE SINCE THE
LAST REPORT

BY
J. M. SMITH
COMMISSIONER

CHICAGO: THE UNIVERSITY OF CHICAGO PRESS
1901

UPTIME(1)

MUNIX

UPTIME(1)

NAME

uptime - show how long system has been up

SYNOPSIS

uptime

DESCRIPTION

Uptime prints the current time and the length of time the system has been up. It is, essentially, the first line of a *w(1)* command.

FILES

/unix system name list

SEE ALSO

w(1)

1947

1947

1947

1947

1947

1947

1947

1947

1947

1947

1947

1947

1947

NAME

users - compact list of users who are on the system

SYNOPSIS

users

DESCRIPTION

Users lists the login names of the users currently on the system in a compact, one-line format.

FILES

/etc/utmp

SEE ALSO

finger(1), who(1)

BUGS

NAME

uucp, *uulog*, *uuname* - unix to unix copy

SYNOPSIS

uucp [option] ... *source-file* ... *destination-file*

uulog [option] ...

uuname

DESCRIPTION

Uucp copies files named by the *source-file* arguments to the *destination-file* argument. A file name may be a path name on your machine, or may have the form:

system-name!path-name

where *system-name* is taken from a list of system names which *uucp* knows about. Shell metacharacters *?*[]* appearing in *path-name* will be expanded on the appropriate system.

Path names may be one of:

- (1) a full path name;
- (2) a path name preceded by *~user* where *user* is a login name on the specified system and is replaced by that user's login directory;
- (3) a path name preceded by *~/user* where *user* is a login name on the specified system and is replaced by that user's directory under PUBDIR;
- (4) anything else is prefixed by the current directory.

If the result is an erroneous path name for the remote system the copy will fail. If the *destination-file* is a directory, the last part of the *source-file* name is used.

Uucp preserves execute permissions across the transmission and gives 0666 read and write permissions (see *chmod(2)*).

The following options are interpreted by *uucp*:

- d Make all necessary directories for the file copy (default).
- f Do not make intermediate directories for the file copy.
- c Use the source file when copying out rather than copying the file to the spool directory (default).
- C Copy the source file to the spool directory.
- m Send mail to the requester when the copy is complete.
- nuser
Notify *user* on the remote system that a file was sent.

-esys

Send the *uucp* command to system *sys* to be executed there. (Note - this will only be successful if the remote machine allows the *uucp* command to be executed by */usr/lib/uucp/uuxqt*.)

Uulog maintains a summary log of *uucp* and *uux(1C)* transactions in the file */usr/spool/uucp/LOGFILE* by gathering information from partial log files named */usr/spool/uucp/LOG.*.?*. (These files will only be created if

1914

1914

1914

The following is a list of the names of the persons who have been elected to the office of the President of the United States since the year 1789. The names are arranged in chronological order, and the year of election is given in parentheses. The names are: George Washington (1789), John Adams (1796), Thomas Jefferson (1800), James Madison (1808), James Monroe (1816), John Quincy Adams (1824), Andrew Jackson (1828), Martin Van Buren (1836), William Henry Harrison (1840), Zachary Taylor (1848), Franklin Pierce (1852), James Buchanan (1856), Abraham Lincoln (1860), Andrew Johnson (1865), Ulysses S. Grant (1868), Rutherford B. Hayes (1876), James A. Garfield (1880), Chester A. Arthur (1881), Benjamin Harrison (1889), Grover Cleveland (1893), William McKinley (1897), Theodore Roosevelt (1901), William Howard Taft (1909), Woodrow Wilson (1913), Warren G. Harding (1921), Calvin Coolidge (1925), Herbert Hoover (1929), Franklin D. Roosevelt (1933), Dwight D. Eisenhower (1953), John F. Kennedy (1961), Lyndon B. Johnson (1964), Richard M. Nixon (1969), Gerald R. Ford (1974), Jimmy Carter (1977), Ronald Reagan (1981), George H. W. Bush (1989), Bill Clinton (1993), George W. Bush (2001), Barack Obama (2009), Donald Trump (2017).

the LOGFILE is being used by another process.) It removes the partial log files.

The options cause *uulog* to print logging information:

-ssys

Print information about work involving system *sys*.

-uuser

Print information about work done for the specified *user*.

Uuname lists the uucp names of known systems. The **-l** option returns the local system name.

FILES

/usr/spool/uucp

spool directory

/usr/spool/uucppublic

public directory for receiving and sending (PUBDIR)

/usr/lib/uucp/•

other data and program files

SEE ALSO

mail(1), uux(1C), uuto(1C), uupick(1C).

Uucp Implementation Description by D. A. Nowitz.

WARNING

The domain of remotely accessible files can (and for obvious security reasons, usually should) be severely restricted. You will very likely not be able to fetch files by path name; ask a responsible person on the remote system to send them to you. For the same reasons you will probably not be able to send files to arbitrary path names. As distributed, the remotely accessible files are those whose names begin /usr/spool/uucppublic (equivalent to ~nuucp or just ~).

BUGS

All files received by *uucp* will be owned by *uucp*.

The **-m** option will only work sending files or receiving a single file. (Receiving multiple files specified by special shell characters **?*[]** will not activate the **-m** option.)

...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...

...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...

...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...
...the ... of ...

NAME

`uustat` - `uucp` status inquiry and job control

SYNOPSIS

`uustat` [option] ...

DESCRIPTION

`Uustat` will display the status of, or cancel, previously specified `uucp` commands, or provide general status on `uucp` connections to other systems. The following options are recognized:

- `-mmch` Report the status of accessibility of machine *mch*. If *mch* is specified as *all*, then the status of all machines known to the local `uucp` are provided.
- `-kjobn` Kill the `uucp` request whose job number is *jobn*. The killed `uucp` request must belong to the person issuing the `uustat` command unless he is the super-user.
- `-chour` Remove the status entries which are older than *hour* hours. This administrative option can only be initiated by the user `uucp` or the super-user.
- `-uuser` Report the status of all `uucp` requests issued by *user*.
- `-ssys` Report the status of all `uucp` requests which communicate with remote system *sys*.
- `-ohour` Report the status of all `uucp` requests which are older than *hour* hours.
- `-yhour` Report the status of all `uucp` requests which are younger than *hour* hours.
- `-jall` Report the status of all the `uucp` requests.
- `-v` Report the `uucp` status verbosely. If this option is not specified, a status code is printed with each `uucp` request.

When no options are given, `uustat` outputs the status of all `uucp` requests issued by the current user. Note that only one of the options `-j`, `-m`, `-k`, `-c`, or the rest of other options may be specified.

For example, the command

```
uustat -uhdc -smhtsa -y72 -v
```

will print the verbose status of all `uucp` requests that were issued by user *hdc* to communicate with system *mhtsa* within the last 72 hours. The meanings of the job request status are:

job-number user remote-system command-time status-time status
where the *status* may be either an octal number or a verbose description. The octal code corresponds to the following description:

OCTAL	STATUS
00001	the copy failed, but the reason cannot be determined
00002	permission to access local file is denied
00004	permission to access remote file is denied
00010	bad <code>uucp</code> command is generated
00020	remote system cannot create temporary file
00040	cannot copy to remote directory
00100	cannot copy to local directory
00200	local system cannot create temporary file

00400 cannot execute *uucp*
01000 copy succeeded
02000 copy finished, job deleted
04000 job is queued

The meanings of the machine accessibility status are:

system-name time status

where *time* is the latest status time and *status* is a self-explanatory description of the machine status.

FILES

/usr/spool/uucp
spool directory
/usr/lib/uucp/L_stat
system status file
/usr/lib/uucp/R_stat
request status file

SEE ALSO

uucp(1C).

Uustat - A UUCP Status Inquiry Program, by H. Che.

NAME

uuto, *uupick* - public UNIX-to-UNIX file copy

SYNOPSIS

uuto [options] source-files destination
uupick [-s system]

DESCRIPTION

Uuto sends *source-files* to *destination*. *Uuto* uses the *uucp*(1C) facility to send files, while it allows the local system to control the file access. A source-file name is a path name on your machine. Destination has the form:

system!user

where *system* is taken from a list of system names that *uucp* knows about (see *uname*(1C)). *Logname* is the login name of someone on the specified system.

Two options are available:

- p Copy the source file into the spool directory before transmission.
- m Send mail to the sender when the copy is complete.

The files (or sub-trees if directories are specified) are sent to PUBDIR on *system*, where PUBDIR is a public directory defined in the *uucp* source. Specifically the files are sent to

PUBDIR/receive/user/mysystem/files.

The destined recipient is notified by *mail*(1) of the arrival of files.

Uupick accepts or rejects the files transmitted to the user. Specifically, *uupick* searches PUBDIR for files destined for the user. For each entry (file or directory) found, the following message is printed on the standard output:

from system: [file file-name] [dir dirname] ?

Uupick then reads a line from the standard input to determine the disposition of the file:

- <new-line> Go on to next entry.
- d Delete the entry.
- m [dir] Move the entry to named directory *dir* (current directory is default).
- a [dir] Same as m except moving all the files sent from *system*.
- p Print the content of the file.
- q Stop.
- EOT (control-d) Same as q.
- !command Escape to the shell to do *command*.
- * Print a command summary.

Uupick invoked with the -ssystem option will only search the PUBDIR for files sent from *system*.

FILES

PUBDIR /usr/spool/uucppublic public directory

SEE ALSO

mail(1), uuclean(8), uucp(1C), uulog(1C), uuname(1C), uustat(1C),
uux(1C).

NAME

`uux` - unix to unix command execution

SYNOPSIS

`uux [-] command-string`

DESCRIPTION

Uux will gather zero or more files from various systems, execute a command on a specified system and then send standard output to a file on a specified system. Note that, for security reasons, many installations will limit the list of commands executable on behalf of an incoming request from *uux*. Many sites will permit little more than the receipt of mail (see *mail(1)*) via *uux*.

The *command-string* is made up of one or more arguments that look like a Shell command line, except that the command and file names may be prefixed by *system-name*!. A null *system-name* is interpreted as the local system.

File names may be one of

- (1) a full path name;
- (2) a path name preceded by *~xxx* where *xxx* is a login name on the specified system and is replaced by that user's login directory;
- (3) anything else is prefixed by the current directory.

The `-` option will cause the standard input to the *uux* command to be the standard input to the *command-string*. For example, the command

```
uux '!diff usg!/usr/dan/f1 pwba!/a4/dan/f1 > !f1.diff'
```

will get the *f1* files from the "usg" and "pwba" machines, execute a *diff* command and put the results in *f1.diff* in the local directory.

Any special shell characters such as `< > ; |` should be quoted either by quoting the entire *command-string*, or quoting the special characters as individual arguments.

Uux will attempt to get all files to the execution system. For files which are output files, the file name must be escaped using parentheses. For example, the command

```
uux a!uucp b!/usr/file \ (c!/usr/file\)
```

will send a *uucp* command to system "a" to get */usr/file* from system "b" and send it to system "c".

Uux will notify you if the requested command on the remote system was disallowed. The response comes by remote mail from the remote machine.

FILES

```
/usr/lib/uucp/spool      spool directory
/usr/lib/uucp/*          other data and programs
```

SEE ALSO

uuclean(8), *uucp(1C)*.
Uucp Implementation Description by D. A. Nowitz

BUGS

Only the first command of a shell pipeline may have a *system-name*!. All other commands are executed on the system of the first command.
The use of the shell metacharacter * will probably not do what you want it to do. The shell tokens << and >> are not implemented.

NAME

val - validate SCCS file

SYNOPSIS

val -

val [-s] [-rSID] [-mname] [-ytype] files

DESCRIPTION

Val determines if the specified *file* is an SCCS file meeting the characteristics specified by the optional argument list. Arguments to *val* may appear in any order. The arguments consist of keyletter arguments, which begin with a -, and named files.

Val has a special argument, -, which causes reading of the standard input until an end-of-file condition is detected. Each line read is independently processed as if it were a command line argument list.

Val generates diagnostic messages on the standard output for each command line and file processed and also returns a single 8-bit code upon exit as described below.

The keyletter arguments are defined as follows. The effects of any keyletter argument apply independently to each named file on the command line.

- | | |
|--------|---|
| -s | The presence of this argument silences the diagnostic message normally generated on the standard output for any error that is detected while processing each named file on a given command line. |
| -rSID | The argument value <i>SID</i> (SCCS IDentification String) is an SCCS delta number. A check is made to determine if the <i>SID</i> is ambiguous (e. g., r1 is ambiguous because it physically does not exist but implies 1.1, 1.2, etc. which may exist) or invalid (e. g., r1.0 or r1.1.0 are invalid because neither case can exist as a valid delta number). If the <i>SID</i> is valid and not ambiguous, a check is made to determine if it actually exists. |
| -mname | The argument value <i>name</i> is compared with the SCCS %M% keyword in <i>file</i> . |
| -ytype | The argument value <i>type</i> is compared with the SCCS %Y% keyword in <i>file</i> . |

The 8-bit code returned by *val* is a disjunction of the possible errors, i. e., can be interpreted as a bit string where (moving from left to right) set bits are interpreted as follows:

- bit 0 = missing file argument;
- bit 1 = unknown or duplicate keyletter argument;
- bit 2 = corrupted SCCS file;
- bit 3 = can't open file or file not SCCS;
- bit 4 = *SID* is invalid or ambiguous;
- bit 5 = *SID* does not exist;
- bit 6 = %Y%, -y mismatch;
- bit 7 = %M%, -m mismatch;

Note that *val* can process two or more files on a given command line and in turn can process multiple command lines (when reading the standard input). In these cases an aggregate code is returned — a logical OR of the codes generated for each command line and file processed.

SEE ALSO

admin(1), delta(1), get(1), prs(1).

DIAGNOSTICS

Use *help*(1) for explanations.

BUGS

Val can process up to 50 files on a single command line. Any number above 50 will produce a **core** dump.

NAME

vc - version control

SYNOPSIS

vc [-a] [-t] [-cchar] [-s] [keyword=value ... keyword=value]

DESCRIPTION

The *vc* command copies lines from the standard input to the standard output under control of its *arguments* and *control statements* encountered in the standard input. In the process of performing the copy operation, user declared *keywords* may be replaced by their string *value* when they appear in plain text and/or control statements.

The copying of lines from the standard input to the standard output is conditional, based on tests (in control statements) of keyword values specified in control statements or as *vc* command arguments.

A control statement is a single line beginning with a control character, except as modified by the *-t* keyletter (see below). The default control character is colon (:), except as modified by the *-c* keyletter (see below). Input lines beginning with a backslash (\) followed by a control character are not control lines and are copied to the standard output with the backslash removed. Lines beginning with a backslash followed by a non-control character are copied in their entirety.

A keyword is composed of 9 or less alphanumeric; the first must be alphabetic. A value is any ASCII string that can be created with *ed*(1); a numeric value is an unsigned string of digits. Keyword values may not contain blanks or tabs.

Replacement of keywords by values is done whenever a keyword surrounded by control characters is encountered on a version control statement. The *-a* keyletter (see below) forces replacement of keywords in *all* lines of text. An uninterpreted control character may be included in a value by preceding it with \. If a literal \ is desired, then it too must be preceded by \.

Keyletter arguments

- | | |
|---------------|--|
| <i>-a</i> | Forces replacement of keywords surrounded by control characters with their assigned value in <i>all</i> text lines and not just in <i>vc</i> statements. |
| <i>-t</i> | All characters from the beginning of a line up to and including the first <i>tab</i> character are ignored for the purpose of detecting a control statement. If one is found, all characters up to and including the <i>tab</i> are discarded. |
| <i>-cchar</i> | Specifies a control character to be used in place of :. |
| <i>-s</i> | Silences warning messages (not error) that are normally printed on the diagnostic output. |

Version Control Statements

:dcl keyword[, ..., keyword]

Used to declare keywords. All keywords must be declared.

:asg keyword=value

Used to assign values to keywords. An **asg** statement overrides the assignment for the corresponding keyword on the **vc** command line and all previous **asg**'s for that keyword. Keywords declared, but not assigned values have null values.

:if condition

⋮

:end

Used to skip lines of the standard input. If the condition is true all lines between the **if** statement and the matching **end** statement are copied to the standard output. If the condition is false, all intervening lines are discarded, including control statements. Note that intervening **if** statements and matching **end** statements are recognized solely for the purpose of maintaining the proper **if-end** matching.

The syntax of a condition is:

<cond>	::= ["not"] <or>
<or>	::= <and> <and> "!" <or>
<and>	::= <exp> <exp> "&" <and>
<exp>	::= "(" <or> ")" <value> <op> <value>
<op>	::= "=" "!=" "<" ">"
<value>	::= <arbitrary ASCII string> <numeric string>

The available operators and their meanings are:

=	equal
!=	not equal
&	and
	or
>	greater than
<	less than
()	used for logical groupings
not	may only occur immediately after the if , and when present, inverts the value of the entire condition

The **>** and **<** operate only on unsigned integer values (e. g.: 012 > 12 is false). All other operators take strings as arguments (e. g.: 012 != 12 is true). The precedence of the operators (from highest to lowest) is:

= != > <	all of equal precedence
&	

Parentheses may be used to alter the order of precedence.

Values must be separated from operators or parentheses by at least one blank or tab.

::text

Used for keyword replacement on lines that are copied to the standard output. The two leading control characters are removed, and

keywords surrounded by control characters in text are replaced by their value before the line is copied to the output file. This action is independent of the `-a` keyletter.

:on

:off

Turn on or off keyword replacement on all lines.

:ctl char

Change the control character to char.

:msg message

Prints the given message on the diagnostic output.

:err message

Prints the given message followed by:

ERROR: err statement on line ... (915)
on the diagnostic output. `Vc` halts execution, and returns an exit code of 1.

DIAGNOSTICS

Use `help(1)` for explanations.

EXIT CODES

0 - normal

1 - any error

1. The first part of the document is a letter from the President of the United States to the Congress, dated January 3, 1862. It is a very important document, as it contains the President's views on the state of the Union and the progress of the war.

2. The second part of the document is a report from the Secretary of the War Department, dated January 10, 1862. It contains a detailed account of the military operations of the Army during the year 1861, and a statement of the condition of the Army at the beginning and end of the year.

3. The third part of the document is a report from the Secretary of the Navy Department, dated January 10, 1862. It contains a detailed account of the naval operations of the Navy during the year 1861, and a statement of the condition of the Navy at the beginning and end of the year.

4. The fourth part of the document is a report from the Secretary of the Department of the Interior, dated January 10, 1862. It contains a detailed account of the operations of the Department during the year 1861, and a statement of the condition of the Department at the beginning and end of the year.

NAME

`vfontinfo` - inspect and print out information about fonts

SYNOPSIS

`vfontinfo` [`-v`] *fontname* [*characters*]

DESCRIPTION

Vfontinfo allows you to examine a font in the UNIX format. It prints out all the information in the font header and information about every non-null (width > 0) glyph. This can be used to make sure the font is consistent with the format.

The *fontname* argument is the name of the font you wish to inspect. It writes to standard output. If it can't find the file in your working directory, it looks in `/usr/lib/vfont` (the place most of the fonts are kept).

The *characters*, if given, specify certain characters to show. If omitted, the entire font is shown.

If the `-v` (verbose) flag is used, the bits of the glyph itself are shown as an array of X's and spaces, in addition to the header information.

SEE ALSO

`vfont`(5)
The Berkeley Font Catalog

AUTHORS

Mark Horton
Andy Hertzfeld

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

100-100000

NAME

vi - screen oriented (visual) display editor based on *ex*

SYNOPSIS

vi [*-t tag*] [*-r file*] [*+command*] [*-l*] [*-wn*] [*-x*] *name* ...

DESCRIPTION

Vi (visual) is a display oriented text editor based on an underlying line editor *ex*(1). It is possible to use the command mode of *ex* from within *vi* and vice-versa.

When using *vi* changes you make to the file are reflected in what you see on your terminal screen. The position of the cursor on the screen indicates the position within the file. The *Vi Quick Reference* card and the *Introduction to Display Editing with Vi* provide full details on using *vi*.

INVOCATION

The following invocation options are interpreted by *vi*:

- ttag* Edit the file containing the *tag* and position the editor at its definition.
- rfile* Recover *file* after an editor or system crash. If *file* is not specified a list of all saved files will be printed.
- +command* Begin editing by executing the specified editor search or positioning *command*.
- l* LISP mode; indents appropriately for lisp code, the *() {} []* and *]]* commands in *vi* and *open* are modified to have meaning for *lisp*.
- wn* Set the default window size to *n*. This is useful when using the editor over a slow speed line.
- x* Encryption mode; a key is prompted for allowing creation or editing of an encrypted file.

The *name* argument indicates files to be edited.

"VI STATES"

- Command Normal and initial state. Other states return to command state upon completion. ESC (escape) is used to cancel a partial command.
- Insert Entered by *a i A I o O c C s S R*. Arbitrary text may then be entered. Insert is normally terminated with ESC character, or abnormally with interrupt.
- Last line Reading input for *:* */* *?* or *!*; terminate with ESC or CR to execute, interrupt to cancel.

COMMANDS

Counts before *vi* commands

line/column number	<i>z G </i>
scroll amount	<i>^D ^U</i>
replicate insert	<i>a i A I</i>
repeat effect	most of the rest

Sample commands

<i>dw</i>	delete a word
-----------	---------------

de	... leaving white space
dd	delete a line
3dd	... 3 lines
itextESC	insert text <i>abc</i>
cwnewESC	change word to <i>new</i>
easESC	pluralize word
xp	transpose characters
ZZ	exit vi

Interrupting, canceling

ESC	end insert or incomplete cmd
~?	(delete or rubout) interrupts
~L	reprint screen if ~? scrambles it

File manipulation

:w	write back changes
:wq	write and quit
:q	quit
:q!	quit, discard changes
:e <i>name</i>	edit file <i>name</i>
:e!	reedit, discard changes
:e + <i>name</i>	edit, starting at end
:e + <i>n</i>	edit starting at line <i>n</i>
:e #	edit alternate file
^	synonym for :e #
:w <i>name</i>	write file <i>name</i>
:w! <i>name</i>	overwrite file <i>name</i>
:sh	run shell, then return
:!cmd	run <i>cmd</i> , then return
:n	edit next file in arglist
:n args	specify new arglist
:f	show current file and line
^G	synonym for :f
:ta <i>tag</i>	to tag file entry <i>tag</i>
~]	:ta, following word is <i>tag</i>

Positioning within file

~F	forward screen
~B	backward screen
~D	scroll down half screen
~U	scroll up half screen
G	goto line (end default)
/pat	next line matching <i>pat</i>
?pat	prev line matching <i>pat</i>
n	repeat last / or ?
N	reverse last / or ?
/pat/+n	n'th line after <i>pat</i>
?pat?-n	n'th line before <i>pat</i>
]]	next section/function
[[previous section/function
%	find matching () { or }

Adjusting the screen

~L	clear and redraw
----	------------------

~R	retype, eliminate @ lines
zCR	redraw, current at window top
z-	... at bottom
z.	... at center
/pat/z-	pat line at bottom
zn.	use n line window
~E	scroll window down 1 line
~Y	scroll window up 1 line

Marking and returning

``	previous context
''	... at first non-white in line
mz	mark position with letter z
`z	to mark z
'z	... at first non-white in line

Line positioning

H	home window line
L	last window line
M	middle window line
+	next line, at first non-white
-	previous line, at first non-white
CR	return, same as +
↓ or j	next line, same column
↑ or k	previous line, same column

Character positioning

^	first non white
O	beginning of line
\$	end of line
h or →	forward
l or ←	backwards
~H	same as ←
space	same as →
fz	find z forward
Fz	f backward
tz	upto z forward
Tz	back upto z
:	repeat last f F t or T
.	inverse of ;
	to specified column
%	find matching ({) or }

Words, sentences, paragraphs

w	word forward
b	back word
e	end of word
)	to next sentence
}	to next paragraph
(back sentence
{	back paragraph
W	blank delimited word
B	back W
E	to end of W

Commands for LISP Mode

) Forward s-expression
} ... but don't stop at atoms
(Back s-expression
{ ... but don't stop at atoms

Corrections during insert

~H erase last character
~W erase last word
erase your erase, same as ~H
kill your kill, erase input this line
\ escapes ~H, your erase and kill
ESC ends insertion, back to command
~? interrupt, terminates insert
~D backtab over *autoindent*
↑~D kill *autoindent*, save for next
O~D ... but at margin next also
~V quote non-printing character

Insert and replace

a append after cursor
i insert before
A append at end of line
I insert before first non-blank
o open line below
O open above
rz replace single char with *x*
R replace characters

Operators (double to affect lines)

d delete
c change
< left shift
> right shift
! filter through command
= indent for LISP
y yank lines to buffer

Miscellaneous operations

C change rest of line
D delete rest of line
s substitute chars
S substitute lines
J join lines
x delete characters
X ... before cursor
Y yank lines

Yank and put

p put back lines
P put before
"xp put from buffer *x*
"xy yank to buffer *x*
"xd delete into buffer *x*

Undo, redo, retrieve

u	undo last change
U	restore current line
.	repeat last change
"dp	retrieve d'th last delete

AUTHOR

Vi and ex are based on software developed by The University of California, Berkeley California, Computer Science Division, Department of Electrical Engineering and Computer Science.

SEE ALSO

ex (1). "Vi Quick Reference" card and "An Introduction to Display Editing with Vi", in the UNIX System Document Processing Guide.

CAVEATS AND BUGS

Software tabs using ^T work only immediately after the *autoindent*.

Left and right shifts on intelligent terminals don't make use of insert and delete character operations in the terminal.

The *wrapmargin* option can be fooled since it looks at output columns when blanks are typed. If a long word passes through the margin and onto the next line without a break, then the line won't be broken.

Insert/delete within a line can be slow if tabs are present on intelligent terminals, since the terminals need help in doing this correctly.

Saving text on deletes in the named buffers is somewhat inefficient.

The *source* command does not work when executed as *:source*; there is no way to use the *:append*, *:change*, and *:insert* commands, since it is not possible to give more than one line of input to a *:* escape. To use these on a *:global* you must Q to ex command mode, execute them, and then reenter the screen editor with *vi* or *open*.

1917

1. The first part of the report deals with the general situation of the country and the progress of the war. It is a very interesting and informative account of the events of the year.

2. The second part of the report deals with the financial situation of the country. It shows that the government has been able to maintain a balanced budget throughout the year, which is a very commendable achievement.

3. The third part of the report deals with the social and economic conditions of the country. It shows that the country has made considerable progress in these fields during the year.

4. The fourth part of the report deals with the foreign relations of the country. It shows that the country has maintained a friendly and peaceful relations with all its neighbors.

5. The fifth part of the report deals with the military situation of the country. It shows that the country has a strong and well-trained army, which is capable of defending the country against any aggression.

6. The sixth part of the report deals with the education system of the country. It shows that the government has made considerable progress in improving the education system during the year.

7. The seventh part of the report deals with the health and welfare of the people. It shows that the government has taken effective measures to improve the health and welfare of the people.

8. The eighth part of the report deals with the agriculture and industry of the country. It shows that the country has made considerable progress in these fields during the year.

9. The ninth part of the report deals with the transportation system of the country. It shows that the government has made considerable progress in improving the transportation system during the year.

10. The tenth part of the report deals with the culture and arts of the country. It shows that the country has a rich and diverse culture and arts scene.

NAME

vsh - Visual Shell

PCS Version 3.97

SYNOPSIS

vsh

DESCRIPTION

Vsh is a highly interactive, screen oriented program which simplifies access to the normal Unix shell and the file system and eases many programming activities. Vsh can be used as a front-end interface to the Unix system for program development and normal applications. Most commands involve only a single keystroke. There are commands in vsh that cause directories or text files to be displayed, editors to be called, compilers to be invoked, etc..

On entry to vsh (ie. after typing in the shell command vsh), the user's current directory is plotted on the screen. This is called a directory page. The top line of the directory page displays the directory's name. Each file in the directory is labeled with a letter (a through t for up to twenty files per page (on a normal terminal)).

To select a file, one presses its corresponding letter. What happens after selection of a file depends on the nature (file type) of the file. If the file is a directory, vsh changes to that directory (similar to the shell command cd). For text files vsh calls the user's default editor (set in his/her environment). There are several other cases for different types of files. Full details of this mechanism (automatic file selection) are given later.

vsh displays up to twenty files at a time on a directory page. Directories which are larger are broken into pages of twenty-files each. The top line of the directory page displays the current page number and total number of pages in the working directory along with its name. There can be up to ten pages holding a total of two hundred files per directory. To select a page other than the current one, one can press a digit (0-9). Pages are numbered one through ten with 0 selecting page ten. There are several other commands for selecting pages (<RETURN>, for example) that are described below. In general in vsh there is more than one way to call the most common commands. This redundancy of command assignment can make it easier to learn vsh as well as simplifying the man-machine interface.

A well integrated feature of Vsh is its connection between compilers and editors for program development. To start a compilation, one presses M. Vsh then calls on the Unix utility make. The output of this compilation from make is saved in a special file that one can later review in showfile mode for compile-time error messages.

In showfile mode, vsh numbers each error message or compiler call in the showfile. One can select an error message by typing its line number. After selection of a line containing an error message, vsh reads the line trying to find a file name and a line number to call the text editor with the proper file at the offending line. For most normal compilers (c, f77 etc...) the format of compiler error messages is standardised and vsh can almost always find the filename and line number. One can jump between the editor and the vsh error listing until one is ready for another

compile.

Because **vsh** is unconventional, the best way to understand how it works is by using it. Because **vsh** is menu driven, it is hard to get lost. The escape mechanism to the normal shell is trivial, one only needs to type **<SPACE>** before any shell command. Beginners should find **vsh** easier to learn than the conventional shell.

FILE SELECTION

One selects a file from the current directory page by pressing its corresponding letter (a through t without **<RETURN>**). What happens after selection depends on the nature of the file selected.

File type	Action upon File Selection
Text	Call the user's default text editor (usually vi or med and set in his .profile file). Like command vi file .
Directory	Change to new (sub-) directory (lower in the tree). Like command cd file .
Executable	(a.out format - 411 magic number) Prompt for input and try to execute. Like command file .
Object module	(*o format - 407 magic number) Print the namelist of the object module. Like command nm -ng file .
Core Dump	Prompt for the name of the executable file that caused the core dump and call default debugger (adb). Like command adb a.out core .
Archive Library	(ar or cpio) Print archive directory. Like command ar -vt file .
Special File, Protected or Other	Selection fails. vsh says it doesn't know what to do.

DIRECTORY OPERATIONS

This is a list of the directory commands. No **<RETURN>** is necessary after these commands (!!). Before doing anything else, the user should become familiar with commands which allow him to select files and display directories. In directories larger than twenty files, the user should be able to display each page.

This list contains only the main assignments of the directory commands. The command overview at the end of this document is a complete list of **vsh** operations and their key assignments.

Command	Action
Q	Leave vsh . Return to login shell. N.B. - vsh ignores ~c and ~z as interrupts on purpose and exits only with 'Q'

	or ~d.
?	Display help file. This describes the commands and options of vsh . There are several different help files for the different modes and pages of vsh .
a - t	Select the corresponding file from the current directory page. (See automatic file selection below)
1 - 9	Select page 1-9 of current working directory.
0	Select page 10.
<RETURN>	Select next page.
-	Select previous page.
\	Change to parent (next-higher) directory. (cd ../)
:	Change to home directory.
/	Change to root directory.
<SPACE>	Escape for one command to the normal Unix shell. vsh prompts for a shell command (to be ended by <RETURN>) and uses your default (login) shell program. When the shell is finished, vsh prompts for 'q' to return to the directory page. Whenever the prompt "Press q to quit" comes, vsh is just waiting in case you want to copy something from the screen before the directory page comes back. Any character typed in after this prompt will do.
,	(Comma) Escape to shell until ~z. vsh forks an interactive subshell. The shell prompt comes. Typing ~z (Control-z) to this shell returns to the directory page.
u - z	User-definable extra file selectors. One can specify 6 files for selection that may belong to any directory. With this mechanism one can have 6 files that are selectable from anywhere in the file hierarchy.
3	Shell environment parameter. vsh prompts with 'S' and one can enter the name of any shell environment parameter. It will then be selected as a file. This command means that one can put a line like "w=/usr/include/sys; export w;" in his/her .profile file and select this directory with "\$w<RETURN>". There can be arbitrarily many such shell parameters.

CAPITAL-LETTER COMMANDS

These are the basic **vsh** commands and its most useful features. They can report system status, handle files, display information and alter the environment of **vsh**.

Many commands ask (prompt) for input, a file name, for example. When typing these responses in, just as in the normal shell, typing <BACK-SPACE> erases the last character typed in and typing '~x' erases the entire input line. Typing <RETURN> ends the response to these prompts.

Instead of a filename one can also type in a single lower-case letter ('a' through 't') to select a file from the current directory page. For those commands that present a list (menu) of possible selections, 'C' for example but not 'R', the action is carried out immediately - no <RETURN> is necessary!!

These are convenience commands.

- H** **(Help)** `vsh` asks for a help subject and prints (using the internal paginator described below) the help file for that command, option or language. This uses the utility `help`.
- I** **(display)** `vsh` prompts for a file and displays it as ascii text. This is faster than a selection which calls the editor. Answer the prompt either with a filename or with a single character for selection from the current directory page. (N.B. This makes it hard to use `vsh` in directories with files that have single-character filenames.). The format is the same as that of the `pg` (1) program (see below).
- D** **(Date)** Display date and time on line 24 of the screen.
- P** **(Process status)** Call `/bin/ps`. Report active processes.
- W** **(Who)** Display who is logged in.
- X** **(eXpr)** Evaluate a mathematical expression. This is like a pocket calculator. `vsh` prompts for an expression in the format of the `math` utility and prints the answer if the evaluation succeeds. Normal operators are +, -, *, /, % (modulo), ^ (to the power of), sin(), cos(), atan(), ln(), exp(), floor(), abs(), rand() (returns a random number between zero and its argument) and parentheses can be used.

These are advanced commands.

- F** **(File)** Select file by name. `vsh` prompts for a file name and then selects the specified file (which need not be on the current directory page). This is useful for selecting directories to perform `cd` to somewhere far away (ie. where it is faster to type the complete file name in than it is to use '\ ' and file selection to get there).
- V** **(Vi)** Select a text file for editing. `vsh` prompts for a file name and calls the default text editor on that file. This works even if you change the `vsh` default action for the selection of a text file (which is normally to call the editor). For browsing one often changes this to `more` or `pg`.
- C** **(Create)** Create a new file. `vsh` prompts for a file name and shows a menu of file types (Text, Directory, Data, Copy, Link or Move). One may create text files (with the editor), directories (with `mkdir`) or empty data files (on which zeroes will be written with a given format and size). One may also copy, link and

move from existing files.

- L** (**ls -l**) Long format file listing. Next to each file, **vsh** displays protection information (access rights), alteration date and file size in the same manner as **ll** or **ls -l**.
- K** Long format listing. Next to each file **vsh** prints its idea of the file's type in the same manner as **file**. This is much slower than 'L' because **vsh** has to open and read something from the beginning of each file to be able to determine its type.
- O** (**Options**) The contents of command and parameter tables are displayed. These control the assignments of capital-letters to functions as well as the file selection actions and help cards. **vsh** prompts for input lines which modify parameters or which create, modify or delete commands. Almost all of the commands and **vsh** functions listed here are interactively alterable and users can save their personalised **vsh** environments in their **\$SAVE/.vshrc** or **\$HOME/.vshrc** files for automatic loading upon entry. Typing **q<RETURN>** in Options Mode returns to the directory page.
- T** (**Touch**) **vsh** prompts for a file name and 'touches' the file. This updates the file's date of last alteration for programs that care or that select the newest file in a directory.

INTERNAL PAGINATOR

The commands 'I' and 'H' use a simple paginator just like the one called **pg (1)**. It displays text files one terminal screen at a time and waits for user input after each page. The last line of the screen is a prompt for user input.

The commands are **f** (forwards) to display the next page of the file, **d** (down) to display the next half-page, **b** (back) to redisplay the first page and **q** (quit) to return to the **vsh** directory page. **<RETURN>** means the same as 'f'.

REMOVE MODE

Press 'R' to enter Remove mode.

In Remove mode, selecting a file causes it to be marked for removal with two slashes, **"/ /"** next to its file selection letter. Files selected a second time are unmarked (they toggle, that is).

Pressing **R** a second time causes all marked files to be removed. The following commands are available in Remove mode:

Command	Action
<Return>	Exit Remove mode without removing marked files.
~d	Same as <Return>
a - t	Select corresponding file for removal. (toggle switch)

- Select all files on current page for removal.
- 0-9 Select and display another page. These commands are described earlier in the manual in the section on directory operations. Files selected for removal on one page are not forgotten by moving to another page. The other page selection commands discussed as directory operations are also available. These include ';', '"', '+' and '-' but not <RETURN>.
- ? Display help file for Remove mode.
- R Remove all marked files on all pages and exit remove mode. If **vsh** cannot remove a file, it displays the reason and waits for a response. Press <Return> to continue removing marked files. Press ^d to immediately exit from Remove mode.

MAKE, GREP, AND SHOWFILE

With its interfaces to **make** and **grep**, **vsh** saves output messages (stdout and stderr) from these programs and allows one to examine the results in Showfile mode. From the directory page, the following commands control **make**, **grep**, and showfile mode.

- G (**grep**) **vsh** prompts for a pattern and file names, and then runs **grep** in the background with these parameters. **grep** searches for the given text pattern in all listed files and reports each occurrence with the file name and line number. The results are saved in the file '.grepout' in the current directory and **vsh** beeps the terminal's bell when finished.
- S (**show**) previous **grep**. The output of the last **grep** run (the file '.grepout') is displayed in Showfile mode. Showfile mode commands are described below.
- M (**make**) **vsh** prompts for a target file (ie. the name of an entry in the makefile) and runs **make**. Execution is controlled by the makefile in the current directory. Output (messages from **make** and/or programs called from the **make** process) is both displayed on the screen and saved in the file '.output.' When **make** terminates, **vsh** enters Showfile mode with the file '.output'.
- N Make running in the background. **make** is run but **vsh** does not wait for termination. Output is saved in '.output', but is not displayed on the screen. When **make** terminates, the bell on your terminal rings twice. Use the E command to review the output in '.output'.
- E (**errors**) The results of the last **make** (in file '.output') are displayed in Showfile mode.

SHOWFILE MODE

Showfile mode displays the results of the execution of **make** or **grep**. It is called by the 'E' and 'S' commands or automatically in the course of an 'M' command.

Showfile mode is essentially a special editor, it displays a selected file with the lines numbered. It has commands which allow one to display specific portions of the file. One cannot alter the file being displayed but can command showfile to read and parse a line of the file. It then scans this line and attempts to extract a file name and line number and then calls the editor, starting it at the specified line. The format of the error messages from most compilers is simple for the showfile mode parser to decode. Take for example this message from the C compiler about the file 'expr.c'

```
c0:expr.c:105: stack undefined; func. expr
```

Selecting this line from a showfile page would call the editor with the file expr.c at line 105.

These are the showfile commands. It is not necessary to press <Return> after these commands. When a complete command is sensed, it is run immediately.

Command	Action
~p	(for '~' a line number) Print error file starting at specified line.
~e	Examine error line '~' for a file name and line number. If they are found, run the editor on this file at the specified line. vsh can find (with relatively good accuracy) file names and line numbers for error messages from the c, pascal and f77 compilers, grep messages and shell command errors.
~<RETURN>	Same as ~e
<RETURN>	Display the next twenty lines.
q	Leave Showfile Mode. Return to directory page.
~d	Same as q
?	Display help file for Showfile mode.

All commands from the directory page which might be useful are available in Showfile mode. They include :, /, .., and 'A' through 'Z' except 'L', 'K' and 'R'.

Particularly useful are 'M' and 'G' which allow new iterations of **make** and **grep**, the escapes to the shell, and 'F' and 'V', which allow one to edit files not found in the the error message file being shown.

MENU MODE

Vsh also offers a mode called menu mode which is like showfile mode except that the messages that are displayed are not error messages but shell commands with prompts and comments to explain their effects.

Menu mode can be entered with 'A' and looks for a file named '.menu' in the current directory or the user's home directory. This file should be a list of shell commands with or without prompts and comments. Typical menu mode lines might look like:

```
whereis $;   whereis the binary & doc for this command
wc $;       word count of the contents of an ascii file
set;        show settings of shell parameters
```

One can see the format of these entries. The shell command, exactly as you would have typed it in comes first. If menu mode is to prompt for parameters or completion of the command, a '\$' must appear in the menu line where the prompted-for parameter is to be. After the command, there can be a ';' and the rest of the menu line will be ignored, it can be used for comments, help, explanation, etc..

To use menu mode, one just has to enter it (normally with 'A') and select a line number from the current screen. If the selected line contains a '\$', menu mode will prompt with the command part of the line and wait to read typed in input (until <RETURN>). Command lines without parameters (ie. without '\$'), are executed immediately.

CONTROL COMMANDS

The operation of **vsh** is controled by its parameter and command tables. The contents of these tables can be displayed and altered with the 'O' command. When **vsh** is invoked, it searches for the file named '.vshrc' in your save or home directorys (defines by the shell parameters \$SAVE and \$HOME). Control comands are then read from this file, allowing one to create a personalized environment.

The 'O' command also prompts for control parameters, allowing for interactive modification of the tables. These control parameters should be presented in the same lexical grammar as shell commands. Spaces and tabs separate tokens. Unquoted newlines terminate each statement.

A character may be quoted by preceding it with a \. All-characters inside single quotes (' ') are quoted except the single quote. Inside double quotes (" "), \ quotes another double quote and \newline is ignored. Grave accents (` `) are treated as any other character. Macros (such as \$HOME) are not available in Options mode.

The lexical grammar is compatible with that of the normal Unix shell. The grammar is somewhat different from that of **cs**h. The file **Misc/.vshrc** in the **vsh** source directory contains default parameters.

To change a parameter, type it in the format of its line in the Options listing, that is:

PARAMETER-NAME PARAMETER-VALUE

where **PARAMETER-NAME** can be any off the following:

Parameter	Usage
editor	Preferred editor. Loaded from shell environment parameter \$EDITOR or set in '.vshrc'. See .profile.

make	make program. Default is '/bin/make'.
grep	grep program. Default is '/bin/grep'.
makerror	output file for make. Default is '.output'.
grepout	output file for grep. Default is '.grepout'.
vshhelp	Help card for '?' command. Default is '/usr/lib/vsh/vshhelp'. There is also a german version in '/usr/src/local/help/vsh'.
rmhelp	Help card for '?' in remove mode.
showhelp	Help card for '?' in showfile mode.
pghelp	Help card for '?' in display mode.
debugger	Debugger program. Default is '/bin/adb'.
help	Help program. Default is '/usr/local/help'.
ffile	File-type program for command 'K'. Default is '/usr/lib/vsh/ffile' (a modified version of 'file' that doesn't print the newline after the file type).
ar	Library archiver program. Default is '/bin/ar'.
cpio	Tape format library program. Default is '/bin/cpio'.
who	Who program. Default is '/bin/who'.
ps	Process status program. Default is '/bin/ps'.
bsh	Bourne shell. Default is '/bin/sh'. (Shell parameter \$SHELL)
csh	Cshell. Default is '/usr/ucb/csh'.
root	Root of the user's file system. Default is '/'.
math	Mathematical expression evaluator. Default is '/usr/local/math'. (From F.R.Moore at C.A.R.L. at C.M.E. at U.C.S.D.)
u - z	File names for 6 extra selectable files. The user can assign these in his .vshrc file and access them thereafter with single keystrokes. They are assigned default values by vsh.

To define a command, use the format:

CHARACTER KEYWORD [PARAMETERS ...]

where CHARACTER is the character which calls the new command.

Valid characters are A-Z and ! " # \$ % & ' () : * = ^ _ [] { } < > . , / ?

These are the valid keywords. They correspond to internal functions in vsh

Keyword	Purpose
date	Display date. Usually 'D'.
showerror	Show errors from previous make . Usually 'E'.
showgrep	Show output from previous grep . Usually 'S'.
showmenu	Show menu from file '.menu'. Usually 'A'.
file	Select a file. If parameter is present, the parameter is selected using the normal criteria for automatic file selection. Otherwise, vsh will prompt for a file name. Usually 'F'.
home	Change to home directory. Usually ':' (often also assigned to '!' in the .vshrc file.)
grep	Run grep . vsh will prompt for parameters. Usually 'G'.
wmake	Run make . Wait for termination and enter Showfile mode. Usually 'M'.
fmake	Run make . Do not wait for termination. Usually 'N'.
remove	Enter Remove mode. Usually 'R'.
longlist	Print long format listing of files on the current page. Usually 'L'.
klonglist	Print long format listing with file type on the current page. Usually 'K'.
create	vsh Enter Create Mode. Create mode has its own menu of file types. Usually 'C'.
dollar	Prompt with '\$' and read a shell parameter for file selection. Usually '\$' and '.'.
display	Display the contents of a file. If no parameter is present, vsh will prompt for a file name. This is the vsh internal version of the pg command as described above. Usually 'I'.
options	Display vsh options (parameter and command tables). Accept commands which modify parameters and which create, modify, and delete commands. If options is called with an argument, it will write the current contents on a file named .vshrc.new and EXIT IMMEDIATELY to allow the user to re-enter with the new init. file. Between exiting and re-entering, the user should move (with mv for example) the .vshrc.new file to .vshrc (after possibly saving the old copy). With this each user can save his vsh environment in his own initialisation file. He only has to set a capital letter to 'options xxx' with options mode and then call this letter. It is good to edit the .vshrc.new file before re-entering to remove this setting. Usually 'O'.
exec	Execute a program directly with the exec system call. The first parameter must be the program name.

Directories in the user's default parameter **\$PATH** are successively searched as in the shell. Any other parameters become parameters to the program. If it is necessary to search several directories for programs, to redirect files, to expand macros, or to expand file regular expressions, use the callshell keyword instead.

prexec	Prompt for input and run a command. The prexec (prompt exec) function takes 3 parameters: [1] the prompt that it should print, [2] the pathname (full or not) of the program it should execute and [3] the character 'g' if prexec should wait for <RETURN> when the command is finished before returning to the directory page otherwise some other character other than 'c' which is used for prompting shells.. If the prompt is to contain spaces (ie. if it has more than one word), you must enclose it in quotes ("...").
callshell	If no parameter is present, an interactive shell is spawned. If one parameter is present, it is passed to the shell and run as a command. Do not specify more than one parameter. Instead, enclose the entire shell command in quotes to provide vsh with only one parameter. Usually ''.
vhel	Display the help card for vsh. Usually '?' from the directory page.
version	Tell which version of vsh you are using. Usually '~'.
callshell	Like callshell only it calls the Cshell.
null	Delete command. The command will no longer be active.

DISPLAY PARAMETERS

vsh has several switches that control the display of the directory page. These are different from parameters and keywords in that they have a specific set of possible values that are listed below.

l	This switch can be set with values 'on' or 'off' and controls the display of the directory page. With l 'on', vsh will display filenames in the format of the ls command, that is, there can be a character after the filename that denotes the file type. Directories are followed by '/', executable files by '*', named pipes by ' ' and special files by 'l'. Other file types are not noted. This option makes the directory display somewhat slower.
t	Normally vsh sorts directories alphabetically by name. When switch t is set 'on' they will be sorted as with the ls -t command, that is by date of reference with the most recently referenced files coming first. This switch can also only take the values 'on' and 'off'.
ncol	vsh can display up to 3 directory columns (up to 60 files) on one directory page. The switch ncol (number of columns) can be set to 1, 2 or 3 to alter this attribute. Note that only files from the

first (left-most) column can be selected and that successive columns are referenced with page number and selection letter.

prompt This switch (when 'on') will cause **vsh** to display a prompt for file selection (or ? for help) on each directory page.

ENVIRONMENT PARAMETERS

vsh takes the following parameters from the global environment:

HOME	The home directory. This parameter is automatically set by Unix when one logs in.
SAVE	The directory to search for '.vshrc'. If \$SAVE is not set, vsh will search in \$HOME . It is recommended that users define \$SAVE as something like ' \$HOME/.save ' and move .profile, .vshrc, .cshrc, etc. there.
SHELL	The users login shell. Unix also sets this parameter. When it is necessary to escape to a shell, this parameter selects the program to use.
TERM	The terminal type. The terminal must be cursor addressable. The initialisation phase of vsh will use this to get the right cursor-positioning commands to work on your current terminal so this better be the terminal type you're really at!

See the Termcap documentation for more information about the **TERM** parameter. Other exported shell parameters can be read by **vsh** for automatic file selection with '\$' or '.' as described above.

MAIL The user's mail file name, usually /usr/mail/user_name. **vsh** checks if there is new mail upon each return from a subprocess and will print the message "you have mail" if there is.

FILES

These are the files **vsh** needs and/or uses.

\$SAVE/.vshrc	(or \$HOME/.vshrc) Initialization file (optional).
/etc/termcap	Terminal capability database necessary for addressing cursor.
.output	File used to save the output of make . This file is always created in the current directory.
.grepout	File used to save the output of grep . vsh tries to first create this file in the working directory. If unsuccessful, the home directory is tried.
.menu	File used for menu mode text. This is also first searched for in the current directory, then in the user's home directory.
/dev/null	Null file.

vsh also calls the user's default shell, editor, make, grep, file, help, math...

EXAMPLE of a '.vshrc' file

This is an example of an initialisation file for vsh. It sets the characters u through z and reassigns several Capital Letters. After calling **vsh** I can perform **cd /usr/src/local/win/lib** with **y** or **rm core** with **B**. Note also that redundant assignments are possible (ie. **!** is now the same as **:**) and that (as in **P**) arguments to commands are possible.

```

u file /usr
v file /usr/sar/stp/net
w file /usr/include/sys
x file /usr/src/local
y file /usr/src/local/win/lib
z file /usr/src/local/buch
B exec /bin/rm core
J exec /usr/bin/readnews
P exec /bin/ps a
T prexec 'whereis : ' /usr/ucb/whereis x
Y exec /bin/df
Z exec /bin/mail
! home
editor /usr/ucb/vi
grep /bin/egrep
llist on
ncol 1

```

The default settings for u through z are:

```

u file /usr
v file /etc
w file /usr/include
x file /lib
y file /usr/lib
z file /usr/src

```

As mentioned above, one can also place lines like:

```
i=/usr/include; export i;
```

in his/her '.profile' file and later select this file with '**\$i<RETURN>**'.

Authors

The history of **vsh** is indeed colorful. This version is based on the so-called 'visual interpreter' that was written by Dave Scheibelhut as part of his thesis at U.C. Berkeley in about 1978. It was distributed as part of UCB's 2.8 BSD PDP-11 Unix tape but later discontinued because of bugs and design problems it contained. This version was stripped down, fixed and ported to a PDP-11 at CMRS at the University of Salzburg by Stephen Pope and further there developed between 1980 and 1983. Version 4.0 was brought onto the PCS Machines and Munix 1.5 System V by STP and

Dittmar Krall at PCS GmbH n Munich, Germany.

BUGS

Vsh has no bugs or inconsistencies whatsoever.

It can get confused, however, by inappropriate input or by typing lots when it expects a little (ie. typing shell commands in without preceding them with <SPACE>). When this happens, typing ^C or <BREAK> will clear the input stream and restore some degree of order.

In directories with more than 200 files, **vsh** ignores files after the 200th. This means the 200th in the unsorted directory list, not necessarily the last alphabetically.

SEE ALSO

Unix Programmers' Manual - Vol 1 - pg(1)
Unix Programmers' Manual - Vol 1 - sh(1)
· Unix Programmers' Manual - Vol 1 - csh(1)
Unix Programmers' Manual - Vol 1 - termcap(3)
Unix Programmers' Manual - Vol 1 - curses(3)
Handbuch fuer das Unix Betriebssystem
Stephen T. Pope, CMRS / PCS, 1983
Chapter 8 is a long tutorial for **vsh** (in German).

Introduction to the Mshell,
Notes on Using the Mshell &
vsh/msh Version 3.8 Distribution Notes
Stephen T. Pope, CMRS,
Cooperative fuer Computermusik,
Universitaet Salzburg, 1981-83

COMMAND SUMMARY

This is a copy of the vsh help card for the directory page.

- Vshell - Command Summary page 1 of 4

Directory Page Commands - No <RETURN> necessary

Command	Action
Q or ^d	- Leave Vsh ('Q' or control-d) - return to login shell
? -	Display this help card - 4 pages - use format of 'I' or pg
a to t-	Select associated file from directory page by letter File selection possibilities described on page 3.
1-9, 0-	Select page 1-9 within current directory, 0 selects page 10.
- or ' -	Select previous page
+ or :-	Select next page
<RETURN>	Select next page
<SPACE> -	Execute temporary shell - return to vsh after 1 command
. or #-	Execute shell - to return to vsh, type ^z to shell
: or ^-	Change to Home directory
\ or] -	Change to previous (parent) directory.
/ -	Change to root directory

Capital Letter Commands - No <RETURN> necessary page 2 of 4

Command	Action
A	Show menu of Shell commands - enter showfile mode
C	Create file or directory - prompt for name & type
D	Date and time
E	Show Errors from previous make - enter showfile mode read file .output in current directory
F	File selection by name - prompt for name
G	Grep - search for a pattern - prompt for pattern & files write file .grepout
H	Help - paginate online helpcard - prompt for subject
I	Display file (text) - prompt for name or letter use internal paginator like pg
K	Long listing - gives file type like the shell command 'file'
L	Long listing - gives date of alteration like ll
M	Make - read makefile - prompt for entry's name write .output - enter showfile mode
N	Make in background (forked off) - prompt for entry's name write .output - beep when finished
O	Options - enter option mode
P	Process status - call ps
R	Remove files - enter remove mode

More Capital Letter Commands - No <RETURN> page 3 of 4

S	Show output of Grep	- enter showfile mode read file .grepout
T	Touch - update a file	- prompt for name or letter
U	Print file - call lpr	- prompt for name or letter
V	edit (vi or med)	- prompt for name or letter
W	Who is logged on - call who	
X	Evaluate mathematical expression-	prompt for expression

File Selection

occurs automatically upon pressing a selection letter in the current dir.

File type selected - Action taken by vsh

Text	-	Execute editor on file. (vi or med) Exiting the editor returns to the same page.
Executable	-	Prompt for input (argv) and execute file. (a.out) Interactive programs take over the terminal.
Directory	-	Change to lower (sub-) directory. (cd)
Archive	-	List contents of archive, ar or cpio format. (ar vt)
Core dump	-	Prompt for object file & debug dump. (adb core)
Other	-	Return error message

Other Vsh Pages

page 4 of 4

have their own help cards viewable with '?' from within each page.

Display Page - Use 'I' for internal paginator - like pg

Commands:

f	- (forwards) - display next 21 lines of file
<RETURN>	same as 'f'
d	- (down) - display next 15 lines
b	- (back) - go to start of file
q	- (quit) - return to directory page

Showfile Page

Commands:

~e	- (edit) search line '~' for a file name and line number, then call editor to that point
<RETURN>	same as ~e
~p	- (print) print showfile starting at line '~'
q	- (quit) - return to directory page

Display mode commands (b,d,f) work in Showfile mode

Remove Page - Use 'R' then select files a through t (or *) from any page.
Press 'R' a second time to perform removal.

Options Page - Use 'O' to view or set vsh internal parameters.
To change one, just type its name and new value, then
<RETURN>. Type q<RETURN> to return to directory page.

Menu Page - Use 'A' to enter menu mode - select comands from the menu
with their line numbers. This is akin to Showfile mode.

NAME

w - who is on and what they are doing

SYNOPSIS

w [-h] [-s] [user]

DESCRIPTION

W prints a summary of the current activity on the system, including what each user is doing. The heading line shows the current time of day, how long the system has been up, the number of users logged into the system, and the load averages. The load average numbers give the number of jobs in the run queue averaged over 1, 5 and 15 minutes.

The fields output are: the users login name, the name of the tty the user is on, the time of day the user logged on, the number of minutes since the user last typed anything, the CPU time used by all processes and their children on that terminal, the CPU time used by the currently active processes, the name and arguments of the current process.

The -h flag suppresses the heading. The -s flag asks for a short form of output. In the short form, the tty is abbreviated, the login time and cpu times are left off, as are the arguments to commands. -l gives the long output, which is the default.

If a user name is included, the output will be restricted to that user.

FILES

/etc/utmp
/dev/kmem
/dev/drum

SEE ALSO

who(1), finger(1), ps(1)

AUTHOR

Mark Horton

BUGS

The notion of the "current process" is muddy. The current algorithm is "the highest numbered process on the terminal that is not ignoring interrupts, or, if there is none, the highest numbered process on the terminal". This fails, for example, in critical sections of programs like the shell and editor, or when faulty programs running in the background fork and fail to ignore interrupts. (In cases where no process can be found, w prints "-".)

The CPU time is only an estimate, in particular, if someone leaves a background process running after logging out, the person currently on that terminal is "charged" with the time.

Background processes are not shown, even though they account for much of the load on the system.

Sometimes processes, typically those in the background, are printed with null or garbaged arguments. In these cases, the name of the command is printed in parentheses.

W does not know about the new conventions for detection of background jobs. It will sometimes find a background job instead of the right one.

NAME

wait - await completion of process

SYNOPSIS

wait

DESCRIPTION

Wait until all processes started with **&** have completed, and report on abnormal terminations.

Because the *wait*(2) system call must be executed in the parent process, the Shell itself executes *wait*, without creating a new process.

SEE ALSO

sh(1)

BUGS

Not all the processes of a 3- or more-stage pipeline are children of the Shell, and thus can't be waited for.

NAME

wc - word count

SYNOPSIS

wc [-lwc] [name ...]

DESCRIPTION

Wc counts lines, words and characters in the named files, or in the standard input if no name appears. A word is a maximal string of characters delimited by spaces, tabs or newlines.

If the optional argument is present, just the specified counts (lines, words or characters) are selected by the letters l, w, or c.

THE UNIVERSITY OF CHICAGO
LIBRARY
540 EAST 57TH STREET
CHICAGO, ILL. 60637

NAME

what - identify SCCS files

SYNOPSIS

what files

DESCRIPTION

What searches the given files for all occurrences of the pattern that *get(1)* substitutes for %Z% (this is @(#) at this printing) and prints out what follows until the first ", >, new-line, \, or null character. For example, if the C program in file *f.c* contains

```
char ident[] = "@(#)identification information";
```

and *f.c* is compiled to yield *f.o* and *a.out*, then the command

```
what f.c f.o a.out
```

will print

```
f.c:      identification information
```

```
f.o:      identification information
```

```
a.out:    identification information
```

What is intended to be used in conjunction with the command *get(1)*, which automatically inserts identifying information, but it can also be used where the information is inserted manually.

SEE ALSO

get(1), *help(1)*.

DIAGNOSTICS

Use *help(1)* for explanations.

BUGS

It's possible that an unintended occurrence of the pattern @(#) could be found just by chance, but this causes no harm in nearly all cases.

10-10-10

10-10-10

10-10-10

The first part of the report is a general description of the project. It includes a brief history of the project, a statement of the problem, and a description of the objectives of the project. The second part of the report is a detailed description of the methodology used in the project. It includes a description of the data collection methods, a description of the data analysis methods, and a description of the results of the project. The third part of the report is a discussion of the results of the project. It includes a discussion of the findings of the project, a discussion of the implications of the findings, and a discussion of the conclusions of the project.

The first part of the report is a general description of the project. It includes a brief history of the project, a statement of the problem, and a description of the objectives of the project. The second part of the report is a detailed description of the methodology used in the project. It includes a description of the data collection methods, a description of the data analysis methods, and a description of the results of the project. The third part of the report is a discussion of the results of the project. It includes a discussion of the findings of the project, a discussion of the implications of the findings, and a discussion of the conclusions of the project.

The first part of the report is a general description of the project. It includes a brief history of the project, a statement of the problem, and a description of the objectives of the project. The second part of the report is a detailed description of the methodology used in the project. It includes a description of the data collection methods, a description of the data analysis methods, and a description of the results of the project. The third part of the report is a discussion of the results of the project. It includes a discussion of the findings of the project, a discussion of the implications of the findings, and a discussion of the conclusions of the project.

10-10-10

NAME

whatis - describe what a command is

SYNOPSIS

whatis command ...

DESCRIPTION

Whatis looks up a given command and gives the header line from the manual section. You can then run the *man*(1) command to get more information. If the line starts 'name(section) ...' you can do 'man section name' to get the documentation for it. Try 'whatis ed' and then you should do 'man 1 ed' to get the manual.

Whatis is actually just the *-f* option to the *man*(1) command.

FILES

/usr/lib/whatis Data base

SEE ALSO

apropos(1), man(1), catman(8)

AUTHOR

William Joy

12574

12574

12574

12574

12574

12574

12574

12574

12574

NAME

whereis - locate source, binary, and or manual for program

SYNOPSIS

whereis [**-sbm**] [**-u**] [**-SBM** dir ... **-f**] name ...

DESCRIPTION

Whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form ".ext", e.g. ".c". Prefixes of "s." resulting from use of source code control are also dealt with. *Whereis* then attempts to locate the desired program in a list of standard places.

You can change this list of standard places. There is a file "/usr/lib/whereis.dirs" which you can edit. Beware of the special line format used there.

If any of the **-b**, **-s** or **-m** flags are given then *whereis* searches only for binaries, sources or manual sections respectively (or any two thereof). The **-u** flag may be used to search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus "whereis -m -u *" asks for those files in the current directory which have no documentation.

Finally, the **-B -M** and **-S** flags may be used to change or otherwise limit the places where *whereis* searches. The **-f** flag is used to terminate the last such directory list and signal the start of file names.

EXAMPLE

The following finds all the files in /usr/bin which are not documented in /usr/man/man1 with source in /usr/src/cmd:

```
cd /usr/ucb
whereis -u -M /usr/man/man1 -S /usr/src/cmd -f *
```

FILES

```
/usr/src/*
/usr/{doc,man}/*
/usr/lib/whereis.dirs
/lib, /etc, /usr/{lib,bin,ucb,old,new,local}
```

AUTHOR

William Joy

BUGS

Since the program uses *chdir*(2) to run faster, pathnames given with the **-M -S** and **-B** must be full; i.e. they must begin with a "/".

NAME

`which` - locate a program file including aliases and paths (*cs*h only)

SYNOPSIS

`which [name] ...`

DESCRIPTION

Which takes a list of names and looks for the files which would be executed had these names been given as commands. Each argument is expanded if it is aliased, and searched for along the user's path. Both aliases and path are taken from the user's `.cshrc` file.

FILES

`~/.cshrc` source of aliases and path values

DIAGNOSTICS

A diagnostic is given for names which are aliased to more than a single word, or if an executable file with the argument name was not found in the path.

BUGS

Only aliases and paths from `~/.cshrc` are used, importing from the current environment is not attempted. Must be executed by a *cs*h, since only *cs*h's know about aliases.

1. The first part of the report is a general introduction to the subject of the study. It discusses the importance of the study and the objectives of the research. It also provides a brief overview of the methodology used in the study.

2. The second part of the report is a detailed description of the study. It includes a description of the sample, the data collection methods, and the analysis techniques used. It also discusses the results of the study and the conclusions drawn from the data.

3. The third part of the report is a discussion of the findings of the study. It compares the results of the study with previous research and discusses the implications of the findings. It also provides recommendations for future research.

4. The fourth part of the report is a conclusion. It summarizes the main findings of the study and provides a final statement on the importance of the research.

NAME

who - who is on the system

SYNOPSIS

who [-uTlpdbrtas] [file]

who am i

DESCRIPTION

Who can list the user's name, terminal line, login time, elapsed time since activity occurred on the line, and the process-ID of the command interpreter (shell) for each current UNIX System user. It examines the */etc/utmp* file to obtain its information. If *file* is given, that file is examined. Usually, *file* will be */etc/wtmp*, which contains a history of all the logins since the file was last created.

Who with the *am i* option identifies the invoking user.

Except for the default *-s* option, the general format for output entries is:

name [state] line time activity pid [comment] [exit]

With options, *who* can list logins, logoffs, reboots, and changes to the system clock, as well as other processes spawned by the *init* process. These options are:

- u This option lists information about those users who are currently logged in. The *name* is the user's login name. The *line* is the name of the line as found in the directory */dev*. The *time* is the time that the user logged in. The *activity* is the number of hours and minutes since activity last occurred on that particular line. A dot (.) indicates that the terminal has seen activity in the last minute and is therefore "current". If more than twenty-four hours have elapsed or the line has not been used since boot time, the entry is marked old. This field is useful when trying to determine whether a person is working at the terminal or not. The *pid* is the process-ID of the user's shell. The *comment* is the comment field associated with this line as found in */etc/inittab* (see *inittab*(5)). This can contain information about where the terminal is located, the telephone number of the dataset, type of terminal if hard-wired, etc.
- T This option causes the *state* of the terminal line to be printed. The *state* describes whether someone else can write to that terminal. A + appears if the terminal is writable by anyone; a - appears if it is not. Root can write to all lines having a + or a - in the *state* field. If a bad line is encountered, a ? is printed.
- l This option lists only those lines on which the system is waiting for someone to login. The *name* field is LOGIN in such cases. Other fields are the same as for user entries except that the *state* field doesn't exist.
- p This option lists any other process which is currently active and has been previously spawned by *init*. The *name* field is the name of the program executed by *init* as found in */etc/inittab*. The *state*, *line*, and *activity* fields have no meaning. The *comment* field shows the *id* field of the line from */etc/inittab* that spawned this process. See *inittab*(5).

- d This option displays all processes that have expired and not been respawned by *init*. The *exit* field appears for dead processes and contains the termination and exit values (as returned by *wait(2)*), of the dead process. This can be useful in determining why a process terminated.
- b This option indicates the time and date of the last reboot.
- r This option indicates the current *run-level* of the *init* process. Following the run-level and date information are three fields which indicate the current state, the number of times that state was previously entered, and the previous state.
- t This option indicates the last change to the system clock (via the *date(1)* command) by *root*. See *su(1)*.
- a This option processes */etc/utmp* or the named *file* with all options turned on.
- s This option is the default and lists only the *name*, *line* and *time* fields.

FILES

/etc/utmp
/etc/wtmp
/etc/inittab

SEE ALSO

date(1), *login(1)*, *mesg(1)*, *su(1)*, *wait(2)*, *inittab(5)*, *utmp(5)*, *init(8)*.

NAME

write - write to another user

SYNOPSIS

write user [line]

DESCRIPTION

Write copies lines from your terminal to that of another user. When first called, it sends the message:

Message from yourname (tty??) [date]...

to the person you want to talk to. When it has successfully completed the connection it also sends two bells to your own terminal to indicate that what you are typing is being sent.

The recipient of the message should write back at this point. Communication continues until an end of file is read from the terminal or an interrupt is sent. At that point *write* writes EOT on the other terminal and exits.

If you want to write to a user who is logged in more than once, the *line* argument may be used to indicate which line or terminal to send to (e.g., *tty00*); otherwise, the first instance of the user found in */etc/utmp* is assumed and the following message posted:

user is logged on more than one place.

You are connected to "*terminal*".

Other locations are:

terminal

Permission to write may be denied or granted by use of the *mesg(1)* command. Writing to others is normally allowed by default. Certain commands, in particular *nroff(1)* and *pr(1)* disallow messages in order to prevent interference with their output. However, if the user has super-user permissions, messages can be forced onto a write inhibited terminal.

If the character **!** is found at the beginning of a line, *write* calls the shell to execute the rest of the line as a command.

The following protocol is suggested for using *write*: when you first *write* to another user, wait for them to *write* back before starting to send. Each person should end a message with a distinctive signal (i.e., **(o)** for "over") so that the other person knows when to reply. The signal **(oo)** (for "over and out") is suggested when conversation is to be terminated.

FILES

/etc/utmp to find user
/bin/sh to execute !

SEE ALSO

mail(1), *mesg(1)*, *nroff(1)*, *pr(1)*, *sh(1)*, *who(1)*.

DIAGNOSTICS

"*user not logged in*" if the person you are trying to *write* to is not logged in.

... ..
... ..
... ..
... ..

... ..
... ..
... ..
... ..
... ..

... ..
... ..
... ..
... ..
... ..

... ..
... ..
... ..
... ..
... ..
... ..
... ..
... ..
... ..
... ..

... ..
... ..

... ..
... ..

... ..
... ..

... ..
... ..

NAME

`xargs` - construct argument list(s) and execute command

SYNOPSIS

`xargs` [flags] [command [initial-arguments]]

DESCRIPTION

`Xargs` combines the fixed *initial-arguments* with arguments read from standard input to execute the specified *command* one or more times. The number of arguments read for each *command* invocation and the manner in which they are combined are determined by the flags specified.

Command, which may be a shell file, is searched for, using one's `$PATH`. If *command* is omitted, `/bin/echo` is used.

Arguments read in from standard input are defined to be contiguous strings of characters delimited by one or more blanks, tabs, or new-lines; empty lines are always discarded. Blanks and tabs may be embedded as part of an argument if escaped or quoted: Characters enclosed in quotes (single or double) are taken literally, and the delimiting quotes are removed. Outside of quoted strings a backslash (\) will escape the next character.

Each argument list is constructed starting with the *initial-arguments*, followed by some number of arguments read from standard input (Exception: see `-i` flag). Flags `-i`, `-l`, and `-n` determine how arguments are selected for each command invocation. When none of these flags are coded, the *initial-arguments* are followed by arguments read continuously from standard input until an internal buffer is full, and then *command* is executed with the accumulated args. This process is repeated until there are no more args. When there are flag conflicts (e.g., `-l` vs. `-n`), the last flag has precedence. Flag values are:

`-lnumber` *Command* is executed for each non-empty *number* lines of arguments from standard input. The last invocation of *command* will be with fewer lines of arguments if fewer than *number* remain. A line is considered to end with the first new-line *unless* the last character of the line is a blank or a tab; a trailing blank/tab signals continuation through the next non-empty line. If *number* is omitted 1 is assumed. Option `-x` is forced.

`-ireplstr` Insert mode: *command* is executed for each line from standard input, taking the entire line as a single arg, inserting it in *initial-arguments* for each occurrence of *replstr*. A maximum of 5 arguments in *initial-arguments* may each contain one or more instances of *replstr*. Blanks and tabs at the beginning of each line are thrown away. Constructed arguments may not grow larger than 255 characters, and option `-x` is also forced. {} is assumed for *replstr* if not specified.

`-nnumber` Execute *command* using as many standard input arguments as possible, up to *number* arguments maximum. Fewer arguments will be used if their total size is

greater than *size* characters, and for the last invocation if there are fewer than *number* arguments remaining. If option *-x* is also coded, each *number* arguments must fit in the *size* limitation, else *xargs* terminates execution.

- t* Trace mode: The *command* and each constructed argument list are echoed to file descriptor 2 just prior to their execution.
- p* Prompt mode: The user is asked whether to execute *command* each invocation. Trace mode (*-t*) is turned on to print the command instance to be executed, followed by a *?... prompt*. A reply of *y* (optionally followed by anything) will execute the command; anything else, including just a carriage return, skips that particular invocation of *command*.
- x* Causes *xargs* to terminate if any argument list would be greater than *size* characters; *-x* is forced by the options *-i* and *-l*. When neither of the options *-i*, *-l*, or *-n* are coded, the total length of all arguments must be within the *size* limit.
- ssize* The maximum total size of each argument list is set to *size* characters; *size* must be a positive integer less than or equal to 470. If *-s* is not coded, 470 is taken as the default. Note that the character count for *size* includes one extra character for each argument and the count of characters in the command name.
- eofstr* *Eofstr* is taken as the logical end-of-file string. Underbar (*_*) is assumed for the logical EOF string if *-e* is not coded. *-e* with no *eofstr* coded turns off the logical EOF string capability (underbar is taken literally). *Xargs* reads standard input until either end-of-file or the logical EOF string is encountered.

Xargs will terminate if either it receives a return code of *-1* from, or if it cannot execute, *command*. When *command* is a shell program, it should explicitly *exit* (see *sh(1)*) with an appropriate value to avoid accidentally returning with *-1*.

EXAMPLES

The following will move all files from directory *\$1* to directory *\$2*, and echo each move command just before doing it:

```
ls $1 | xargs -i -t mv $1/{} $2/{} 
```

The following will combine the output of the parenthesized commands onto one line, which is then echoed to the end of file *log*:

```
(logname; date; echo $0 $*) | xargs >>log 
```

The user is asked which files in the current directory are to be archived and archives them into *arch* (1.) one at a time, or (2.) many at a time.

1. *ls* | *xargs -p -l ar r arch*
2. *ls* | *xargs -p -l | xargs ar r arch*

XARGS(1)

MUNIX

XARGS(1)

The following will execute *diff*(1) with successive pairs of arguments originally typed as shell arguments:

echo \$* | xargs -n2 diff

DIAGNOSTICS

Self explanatory.

NAME

xd - hexadecimal dump

SYNOPSIS

xd [file] [offset1[.][b] [- [offset2[.][b]]

DESCRIPTION

Xd dumps *file* in hexadecimal and as characters.

The *file* argument specifies which file is to be dumped. If no file argument is specified, the standard input is used.

The *offset* argument specifies the offset in the file where dumping is to commence. This argument is normally interpreted as hexadecimal bytes. If it starts with '0', the offset is interpreted in octal. If '.' is appended, the offset is interpreted in decimal. If 'b' is appended, the offset is interpreted in blocks of 512 bytes. If the file argument is omitted, the offset argument must be preceded '+'.

If no second offset is specified, dumping continues until end-of-file. Otherwise dumping ends *before* the specified endaddress. The interpretation of *offset2* is the same as above.

SEE ALSO

adb(1)

NAME

xref - cross-reference listing

SYNOPSIS

xref [-c] [-p] [-P]

DESCRIPTION

Xref creates a cross-reference listing from the standard input.

It lists on the standard output the alphabetic sorted identifiers followed by the linenumbers in which they appear.

-c C comments will be skipped

-p Pascal comments will be skipped

-P Packed files will be processed. The filenames are prepended to the linenumbers.

Unpacked input files are listed without filenames.

EXAMPLE

pack ack.p fac.p | xref -p -P
outputs:

```
:
var      ack.p    2   fac.p      3
writeln  ack.p   12   14   fac.p   10   12
x        ack.p    2   13      14   14   15
         fac.p    3   11      12   12   13
y        ack.p    2   13      14   14
:
```

SEE ALSO

pack(1).

THE UNIVERSITY OF CHICAGO
DIVISION OF THE PHYSICAL SCIENCES
DEPARTMENT OF CHEMISTRY
530 SOUTH EAST ASIAN AVENUE
CHICAGO, ILLINOIS 60607-7070
TEL: 773/936-5200 FAX: 773/936-5201
WWW: WWW.CHEM.UCHICAGO.EDU

RECEIVED
JAN 11 1994
11 11 11
11 11 11
11 11 11

NAME

`xstr` - extract strings from C programs to implement shared strings

SYNOPSIS

`xstr [-c] [-] [file]`

DESCRIPTION

Xstr maintains a file *strings* into which strings in component parts of a large program are hashed. These strings are replaced with references to this common area. This serves to implement shared constant strings, most useful if they are also read-only.

The command

`xstr -c name`

will extract the strings from the C source in *name*, replacing string references by expressions of the form `(&xstr[number])` for some number. An appropriate declaration of *xstr* is prepended to the file. The resulting C text is placed in the file *x.c*, to then be compiled. The strings from this file are placed in the *strings* data base if they are not there already. Repeated strings and strings which are suffices of existing strings do not cause changes to the data base.

After all components of a large program have been compiled a file *xs.c* declaring the common *xstr* space can be created by a command of the form

`xstr`

This *xs.c* file should then be compiled and loaded with the rest of the program. If possible, the array can be made read-only (shared) saving space and swap overhead.

Xstr can also be used on a single file. A command

`xstr name`

creates files *x.c* and *xs.c* as before, without using or affecting any *strings* file in the same directory.

It may be useful to run *xstr* after the C preprocessor if any macro definitions yield strings or if there is conditional code which contains strings which may not, in fact, be needed. *Xstr* reads from its standard input when the argument '-' is given. An appropriate command sequence for running *xstr* after the C preprocessor is:

```
cc -E name.c | xstr -c -
cc -c x.c
mv x.o name.o
```

Xstr does not touch the file *strings* unless new items are added, thus *make* can avoid remaking *xs.o* unless truly necessary.

FILES

<i>strings</i>	Data base of strings
<i>x.c</i>	Massaged C source
<i>xs.c</i>	C source for definition of array 'xstr'
<i>/tmp/xs*</i>	Temp file when 'xstr name' doesn't touch <i>strings</i>

SEE ALSO

mkstr(1)

AUTHOR

William Joy

BUGS

If a string is a suffix of another string in the data base, but the shorter string is seen first by *xstr* both strings will be placed in the data base, when just placing the longer one there will do.

NAME

yacc - yet another compiler-compiler

SYNOPSIS

yacc [-vd] grammar

DESCRIPTION

Yacc converts a context-free grammar into a set of tables for a simple automaton which executes an LR(1) parsing algorithm. The grammar may be ambiguous; specified precedence rules are used to break ambiguities.

The output file, *y.tab.c*, must be compiled by the C compiler to produce a program *yyparse*. This program must be loaded with the lexical analyzer program, *yylex*, as well as *main* and *yyerror*, an error handling routine. These routines must be supplied by the user; *Lex*(1) is useful for creating lexical analyzers usable by *yacc*.

If the *-v* flag is given, the file *y.output* is prepared, which contains a description of the parsing tables and a report on conflicts generated by ambiguities in the grammar.

If the *-d* flag is used, the file *y.tab.h* is generated with the *define* statements that associate the *yacc*-assigned 'token codes' with the user-declared 'token names'. This allows source files other than *y.tab.c* to access the token codes.

FILES

y.output
y.tab.c
y.tab.h defines for token names
yacc.tmp, *yacc.acts* temporary files
/usr/lib/yaccpar parser prototype for C programs
/usr/lib/liby.a library with default 'main' and 'yyerror'

SEE ALSO

lex(1)
LR Parsing by A. V. Aho and S. C. Johnson, Computing Surveys, June, 1974.
YACC - Yet Another Compiler Compiler by S. C. Johnson.

DIAGNOSTICS

The number of reduce-reduce and shift-reduce conflicts is reported on the standard output; a more detailed report is found in the *y.output* file. Similarly, if some rules are not reachable from the start symbol, this is also reported.

BUGS

Because file names are fixed, at most one *yacc* process can be active in a given directory at a time.

Received of the Treasurer of the
County of [illegible] the sum of [illegible]
for [illegible]

Witness my hand and seal of office
this [illegible] day of [illegible] 1875
at [illegible]

By [illegible]
[illegible]

Attest my hand and seal of office
this [illegible] day of [illegible] 1875
at [illegible]

By [illegible]
[illegible]

YES(1)

MUNIX

YES(1)

NAME

yes - be repetitively affirmative

SYNOPSIS

yes [**expletive**]

DESCRIPTION

Yes repeatedly outputs y, or if **expletive** is given, that is output repeatedly. Termination is by rubout.

EXAMPLE

"yes | fsck" would be the same as "fsck -y".

